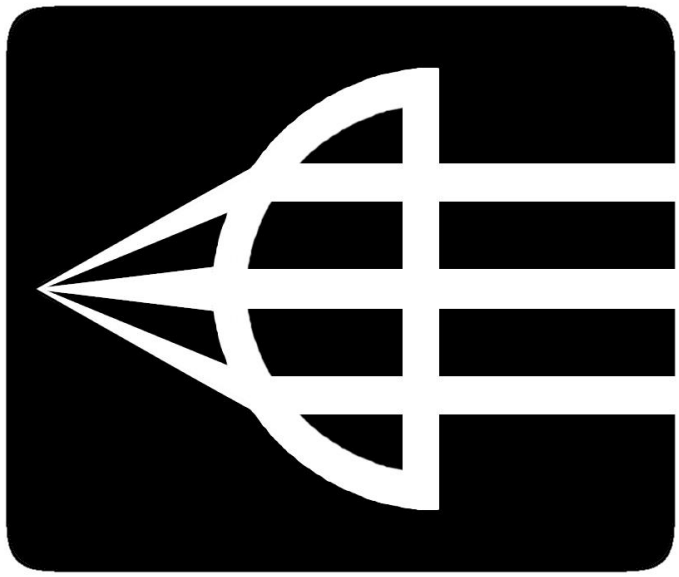
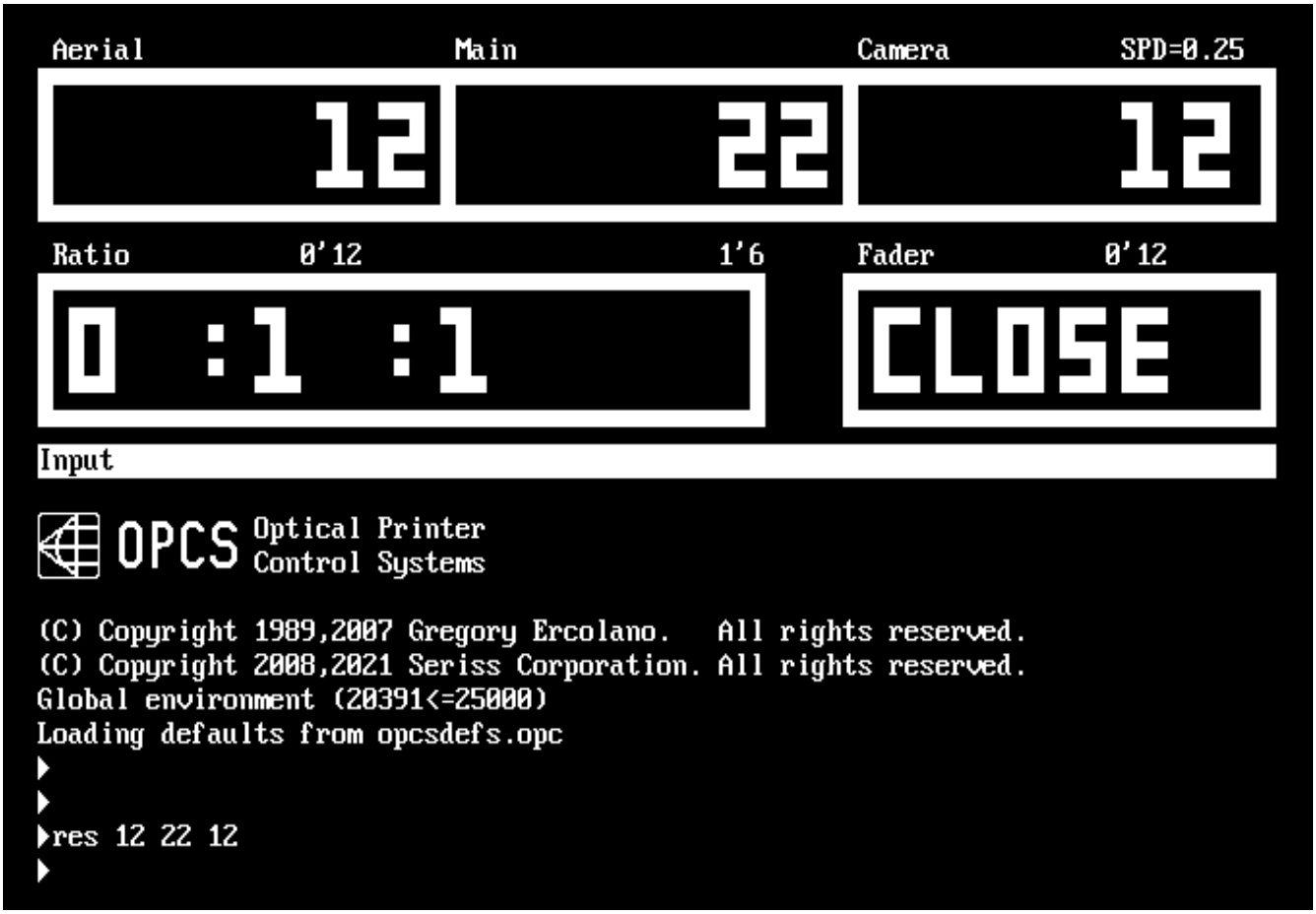


OPTICAL PRINTER CONTROL SYSTEMS
USER'S MANUAL VER. K2.21/TC

June 2024

SERISS
CORPORATION

erco@seriss.com
(626) 576-0010



OPCS screenshot.

OPERATOR'S QUICK REFERENCE

NAME

quickref - OPCS quick reference for camera operators

OPCS QUICK REFERENCE

*** STANDARD PROCEDURES **********

FILM MOVEMENTS

There are two things to remember about the movements that are typical when the OPCS system is idle:

1) The projectors show the SEATED IMAGE that's ABOUT TO BE SHOT

2) The camera is UNSEATED with the shutter closed

This way, you can look through the viewer and see the pinned, seated, in-focus projector image that is about to be shot.

Unexposed film in the camera should be unseated, the camera shutter being closed and not exposing film.

ORDER OF OPERATION

It is important to be familiar with the order of how the system executes shooting operations:

>> 1) Motors seek positions FIRST. (for feeds, fades, dissolves)

>> 2) Camera exposes film SECOND.

>> 3) Projectors advance THIRD.

>> 4) Wedge/filter wheels are moved LAST. (**autofilt**)

If a fade, feed or dissolve is pending, these will move to positions first.

If the camera is told to shoot, it will shoot AFTER feed/fade/dx's have moved to position.

After the camera exposes the image in the projector gate(s), the projector(s) advance to their new positions.

>> Before a step print, remember that the image in the projector's
>> gate is ABOUT TO BE SHOT. The camera always shoots first, and
>> the projectors advance after.

If there are any pending **autofilt** commands, these will move LAST. This way, before a wedge, you are looking at a projector image through the filter wheel position that has no filter in it, making it easier to look through the viewer before shooting a wedge.

ALLSTOP

The allstop key stops running motors at the nearest frame, without affecting exposure, and prompts for ABORT or CONTINUE:

>> CONTINUE: The shot continues shooting as if nothing happened.
>> ABORT: Shooting stops, giving you a new command prompt.

WINDOFF WITH SHUTTER CLOSED

If someone has setup '**seekcap yes**' in your opcsdefs.opc file, the following will cap the shutter and run off the frames at the camera's slewing speed, and then return the fader to its previous position:

```
seek 100        # caps fader, runs off 100 frms on camera at
                # slew speed,  and returns fader to previous
                # position.
```

This is the equivalent, more obvious way, but does not use the 'slew' speed for the camera:

```
cls cam 100 opn     # Close fader, wind cam 100x, open fader
```

STARTING A NEW SHOT

Before starting a shot, you will probably want to reset your counters, roll off some black on the camera, and send the projectors out to their starting frames to prep for the first shot. Here's an actual typical example as typed in by a camera operator:

```
load                  # operator loads camera & projectors
res 0 0 0             # resets all counters to zero
seek >120 >120 >100   # send projectors to start, winds off
                      #  100 frames of black on camera.
rat 1 1 1 spd .25     # Setup a default ratio and camera speed.
opn                   # Make sure shutter is open, and that no
                      # fades or dissolves are pending.
```

The above commands can be put into a file, since it is done each time the operator starts a shot. See RUNCMD(OPCSDEFS) for defining new commands, and RUN(OPCS) for making and executing scripts. An example of making the above into a command would be:

1. Add the following to your OPCSDEFS.OPC file:

   ```
   runcmd newshot newshot.run 2
   ```

2. Enter '**ldefs opcsdefs.opc**' to make sure the RUNCMD takes effect.
3. Make a file called NEWSHOT.RUN with the following commands:

   ```
   load
   res 0 0 0
   seek $1 $2 >100
   rat 1 1 1 spd .25 opn
   ```

You should now be able to type the following at the OPCS prompt, which will automatically run the LOAD command, reset counters, run the projectors out to their starting frames, etc:

```
        newshot >120 >120
```

HOLDS

To hold on a frame currently in the projector's gate, use the CAM command. Note you can specify frames, feet/frames, or absolute frame positions:

```
     cam 12       #  run camera 12 frames
     cam 4'2      #  run camera 4 feet 2 frames
     cam >12      #  run camera TO frame 12 on the counter
     cam >4'2     #  run camera TO 4 feet 2 frames on the counter
```

STRAIGHT PRINTS

To do a straight print, you can do it the slow way, or the fast and efficient way...

```
    do 12  cam 1 pro 1     #  SLOW WAY: straight print 12x
    do 12  cam 1 pro -1    #  SLOW WAY: straight reverse print 12x

    rat 1  1  rep 12       #  FAST WAY: straight print 12x
    rat 1 -1  rep 12       #  FAST WAY: straight reverse print 12x
```

STEP PRINTS

To do a step print of any kind, it is recommended you use RAT and REP:

```
    rat 2 1   rep 12    #  FAST MOTION: 12x step print of every other
                        #  projector frame

    rat 1 2   rep 6     #  SLOW MOTION: 12x step print on twos
                        #  Note REP 6 because camera shoots 2x each time

    rat -1 1  rep 12    #  REVERSE MOTION: 12x reverse print
```

CYCLES

Sometimes it is desirable to cycle projector frames rather than do a simple hold. Here is a back and forth cycle (1,2,3,2,1,2...):

```
    do 12  rat 1 1 rep 3 pro -1 rat -1 1 rep 2
```

Here is a cycle of selected projector frames (3,6,8,6,3...):

```
    do 12  pro >3 cam 1 pro >6 cam 1 pro >8 cam 1 pro >6 cam 1
```

FADES/DISSOLVES
---

1) **fdi 12 cam 12**

2) **rat 1 1 dxo 12 rep 12**

3) **do 12 dxo 4 cam 4 seek 8 -4 dxi 4 cam 4**
^These commands repeat 12 times from left to right.

The first example sets up and shoots a 12x fade-in in on a still projector image.

The second example sets up and shoots a 12x dissolve out of a moving projector image.

The last example does (12) 4-frame cross dissolves on every 8th projector image, having the effect of 'weaving' still frames from an otherwise slow-motion moving projector image. Here's a break down of the command:

do 12 dxo 4 cam 4 seek 8 -4 dxi 4 cam 4

```
  |         |           |           |
  |         |           |           Shoot 4x dissolve in
  |         |           Projector advances 8, camera backwinds 8.
  |         |           (note: fader is closed after DXO 4 CAM 4
  |         |           finished executing, so it's OK to backwind)
  |         Shoot 4x dissolve out.
  Repeat all commands to the right 12 times. One execution would do
  a single cross dissolve. 'do 12' will do 12 cross dissolves.
```

EXPOSURE WEDGES

You can use the SPD command to easily do exposure wedges.
To do a wedge the hard way, you can make a small script out of the following commands:

```
spd .20 cam 1
spd .30 cam 1
spd .40 cam 1
spd .50 cam 1
spd .60 cam 1
spd .70 cam 1
spd .80 cam 1
spd .90 cam 1
spd 1.0 cam 1
```

This shoots a 9 frame wedge of the camera speeds between .20 and 1.0 at .10 increments.

There is a shorthand way to achieve the same, using the 'relative offset' feature in the SPD command:

```
spd .20 do 9 cam 1 spd +.10
```

NOTE: When done executing, the current speed speed will be left at 1.1, even though it won't have been shot as part of the wedge.

FILTER WEDGES

If you have a filter wheel, you can automate wedging the filter wheel using the AUTOFILT command.

For instance, the most efficient way to shoot a simple 20x wedge on the filter wheel:

```
autofilt on  cam 20
```

Or, in long hand, and somewhat slower to shoot:

```
home autofilt         # home the filter wheel
do 20 cam 1 go h 100  # shoot 20x, filter +100 pulses each frm
```

COMBINATION WEDGE

To shoot a combination exposure and filter wedge, you can do the following to wedge both the exposure (1.0 thru 0.2 on .10 increments) and the full 20x on the filter wheel:

```
spd 1.0 do 8 autofilt on cam 20 spd -.1
```

To break this down:

```
spd 1.0      - start the exposure speed at 1.0
do 8         - repeat the following commands 8 times
autofilt on  - home the filter wheel, and enable auto-wedging
cam 20       - shoot 20x; filter wheel will move each frame
spd -.1      - subtract .1 from exposure speed
```

*** AUTOMATED SCRIPTS ***

You can setup scripts to do several operations automatically. Experienced operators can setup a complicated printing operation as a script file, so another operator can shoot it without knowing the intricate details involved.

This also lets you re-run the script later if a reshoot is necessary.

CREATING SCRIPT FILES

You can either use a text editor to create a script of OPCS commands, or you can use the LOG command to have the software save commands to a file as you execute them. If you use the LOG command to create a file, you can use a text editor to make corrections or modifications afterwards.

```
log myfile.log   # Start a command log to the file 'myfile.log'
                 # Commands executed in the OPCS software from
                 # here on will be saved to the file.

log off          # This closes the logfile, and turns off
                 # command logging. Note: if you quit the
                 # OPCS software, it will automatically close
                 # and save any log files that were in progress.
```

EXECUTING SCRIPTS

Once you have a script file, you can run it by executing:

```
    run myfile.log        #  run the commands in the script
```

While the script runs, the filename is shown in the runbar, along with the line number its currently executing.

You may find that at a certain point, you want the script to pause so the operator can change filters, or do some other manual operation before continuing. Use the PSE command for this. Consider this excerpt from a script:

```
    fdo 12 cam 12 cam 24       # fade out and hold black
    seek >1878 -               # find Scene 4A

    # HEY YOU! Wake up, and insert the ND 60 filter for Scene 4A
    @ pse

    rat 2 1 rep 400            # fast motion of guy running
```

When this script executes, the 'HEY YOU' comment will appear on the screen, and PSE will stop with a prompt:

```
    # HEY YOU! Wake up, and insert the ND 60 filter for Scene 4A
    * FILE PAUSE *
    RETURN to continue,  SPACEBAR to abort: _
```

...The operator can then insert the filter, and continue the shot.

While a script is running, the operator can hit ALLSTOP, which will pause the script wherever it happens to be at the time, allowing the option of continuing or aborting.

The operator may decide to ABORT the running script to execute other commands before continuing or to fix a problem. When a script is aborted, the software displays the line number at which the script was stopped. To start the script up from the same place, the operator can supply the 'stopped at' line number to the RUN command:

```
    STOPPED AT LINE 33 IN 'MYSCRIPT.RUN'  # operator aborts script
    run myscript.run 33                   # continue script where left off
```

See the man page on 'run' for more on this.

SCRIPT TIPS

To use scripts effectively, here are some tips:

COMMENTS

Scripts are great when you write them, but you'll never remember what they do days later just by looking at them. 'What the hell did I set this up to do?' comes to mind. PUT COMMENTS IN YOUR SCRIPTS so you (and others) know what it does. A quick one liner comment at the top of your script at least. Use brief comments to describe lines or blocks of lines that you think might need description.

AVOID LONG LINES

Cramming lots of commands needlessly into long lines is best avoided when writing scripts. If nothing else, to make it easy to see what's going on. Also, it makes it easier to restart a script at a particular line.

There's nothing wrong with putting several operations on one line, if they're all related.

MODULARIZE: SMALL SCRIPTS

It's often a good idea to break a long shot into separate scripts, and then make a 'main script' that calls all the smaller ones.

Also, if you find yourself copying blocks of commands over and over into different places in a file, it's probably better to put these commands into a script, and then call that script. This avoids typos, and can simplify things greatly.

KEEP ORIGINALS

If you are going to use a script from another scene, make a copy, and then modify the copy. Never modify your original script if it was used for a shot that may need a reshoot in the future.

SAVE YOUR FILES ON FLOPPIES

Hard disks are great, but when they crash, you can lose the whole works, unless you have backups on floppies. Learn how to format floppies, and copy files to/from them. Keep related shots together on one floppy. You can even make subdirectories on floppies to keep shots separate.

STANDARDIZE RECORD KEEPING

Come up with some sort of standard for keeping track of files for your shots. Since filenames can have up to 8 letters (plus 3 letter extensions), you don't have much flexibility, and can end up with cryptic insanity such as:

```
ILM057T2.RUN                    #  insane file name
|  |  |
|  |  'Take 2'
|  Shot or scene number
3 letter  job name
```

A better technique is to use subdirectories, allowing you the freedom to use any filenames you want (since people tend to make their own names for files anyway):

```
ILM\SHOT05A\DXTEST.RUN          #  first take
ILM\SHOT05A\DXTEST2.RUN         #  second take
ILM\SHOT05A\FINAL.RUN           #  final shot used
 |   |       |
 |   |       Any  filenames  appropriate
 |   Shot  directory
 Job  directory
```

*** MOTION CONTROL MOVES **

The OPCS software supports motion control to the extent that you can either specify moving motors with the GO(OPCS) command, or by an ascii file containing columns of numbers that represent absolute positions using the FEED(OPCS) command. Channels can be moved by hand and 'key frames' can be created using the JOG(OPCS) command.

OPCS does not have any curve editors or graphing programs built into it. However, OPCS does come with some external commands that help you create FEED(OPCS) files. These external commands can be made to look like they're part of OPCS using RUNCMD(OPCSDEFS) and DOSCMD(OPCSDEFS).

You can also generate FEED(OPCS) files by either use existing 3rd party software (like Kuper) or your own custom C programs. FEED files are simple ascii files, which you can even create using a text editor.

OPCS does *NOT* support streaking. See below, STREAKING.

PANS

Pans are relatively straight forward. They usually involve one or more axes that simply move to a position before shooting.

/ DOCUMENTATION IN PROGRESS */*
Show examples of how to create and shoot pans.

ZOOMS

Zooms are somewhat more complicated than simple pans by two peculiarities:

1) Follow focus
2) Exposure compensation

Regarding #1, focus is usually achieved by moving the lens, and having the camera move relative to it. See INTERP(OPCSDEFS) for how to configure auto-focus.

For manual control of focus, you can run the focus channel separately using either FEED(OPCS) or go(OPCS). Simply specifying the focus channel for movement will override auto-focus.

Regarding #2, as you zoom into the film frame, light is lost. This is because the same amount of light for 1:1 spreads out the more you zoom in. Exposure speed can compensate for the lost light by slowing the exposure.

Exposure can either be auto-compensated with SPDINTERP(OPCSDEFS) or by manual specification in a FEED(OPCS) file by specifying a column of numbers to the special channel 'x'.

/ DOCUMENTATION IN PROGRESS */*
Show examples of how to create and shoot zooms.

STREAKING

OPCS does *NOT* support streaking. It never will. This is advertised in the FAQ, and is adamantly underlined.

If you want streaking capabilities, that is a whole other bag of worms which is best handled by dedicated motion control software, such as that supplied by Kuper Controls. OPCS does not purport to be a full on motion control system, and streaking is where the line is drawn.

*** MISCELLANEOUS TIPS **

NEW COMMAND LINE EDITING (K2.00 AND UP)

In OPCS version K2.00 and higher, line editing and a command history
have been added to make it easier to retype and edit commands.

More like a text editor or word processor, you can interactively
edit commands, using LEFT/RIGHT arrow to move back into a long line
to make changes, Delete characters, insert characters, etc:

```
         Up Arrow -- previous line in command history      (^P)
         Dn Arrow -- next line in command history          (^N)
         Lt Arrow -- move reverse one char on current line (^B)
         Rt Arrow -- move forward one char on current line (^F)
       Backspace -- backspace and delete                   (^H)
          Delete -- delete character                       (^D)
            Home -- move to start of current line          (^A)
             End -- move to end of current line            (^E)
       Ctrl-Home -- jump to top of command history
        Ctrl-End -- jump to bottom of command history (current line)
       Ctrl-Left -- word left
      Ctrl-Right -- word right
              ^K -- clear to end of line
              ^U -- clear current line (hit again to 'undo')
              ^V -- enter next character literally
             ESC -- clear current line (hit again to 'undo')
              F3 -- re-type last command
              F4 -- re-run last command (F3 + Enter)
```

OPERATOR PREFERENCES

The following OPCSDEFS.OPC commands can be set up to the
operator's taste. You will not 'mess anything up' if you
change these commands, they are only for the operator to
play with, and will not effect motors, running speeds, etc.

bigcounters [on or off]
If you prefer the large screen counters set 'bigcounters on'.
If you would rather have more screen space to see commands
and files, you may want to try 'bigcounters off'.

leadingzeroes [on or off]
If you don’t like all the leading zeroes in the counter
displays, you can turn them off with 'leadingzeroes off'.

pro2display [on or off]
If you have a single headed printer, you will probably
prefer not to have the unwanted 'projector 2' counter on
the screen. You can turn it off by setting 'pro2display off'.
However, if you have an aerial head, you will want this
setting 'on'.

If you want to change these values permanently, alter the OPCSDEFS.OPC file. If you just want to try them out with out having them be a permanent change, you can use the LDEFS command to read commands from the keyboard:

```
ldefs con              # type this at the OPCS prompt
bigcounters off        # try a different setting
^Z                     # (type control-z and hit return)
                       # Back in OPCS, big counters off.
```

SPECIAL COMMAND USAGE: THE CALCULATOR

At any time in the OPCS software, you can do calculator entries. Simply form a math expression that is encapsulated in parentheses. Do this on an empty OPCS command line. When you hit return, the answer will be displayed:

```
(20*(12+8+9+54))      # type this in to add up your hours..
1660.0000             # Result displayed
```

If you are in DOS, there is a 'calc' command that will let you do the same thing:

```
C:\USR\OPCS>calc        #  run CALC from DOS
(34+sqrt(47))           #  type in a math expression
40.855655               #  Result

(3+4+5)                 #  keep entering commands..
12.000000               #  Result

^C                      #  Type CONTROL-C to return to DOS
C:\USR\OPCS>            #  (back in DOS)
```

These are the math functions allowed currently for math expressions:

```
/*** MATH FUNCTIONS  ***/
sqrt()     log()          exp()
sin()      cos()          tan()
asin()     acos()         atan()
radians()  degrees()

/*** NUMERIC EXPRESSIONS  ***/
-12           #  negative 12
+34           #  positive 34
0x3ff         #  hex representation for 1023 decimal
```

GETTING HELP

Use the MAN command to get general and specific documentation. The '-k' flag tells MAN to be general, and look for anything related to the argument that follows. In the following examples, case is important (i.e. upper/lower case):

*** General Documentation **

```
    man -k OPCS:       # list OPCS commands
    man -k OPCSDEFS:   # list opcsdefs commands
    man -k OPCS        # list OPCS related commands
    man -k             # list everything MAN knows about
```

Any if the above will display a long list of commands, followed by simple one line descriptions of each command. With this list, you can then zero in on any of the commands to get more complete documentation...

*** Detailed Documentation ***************************************

```
    man cam            #  specific docs on the CAM command
```

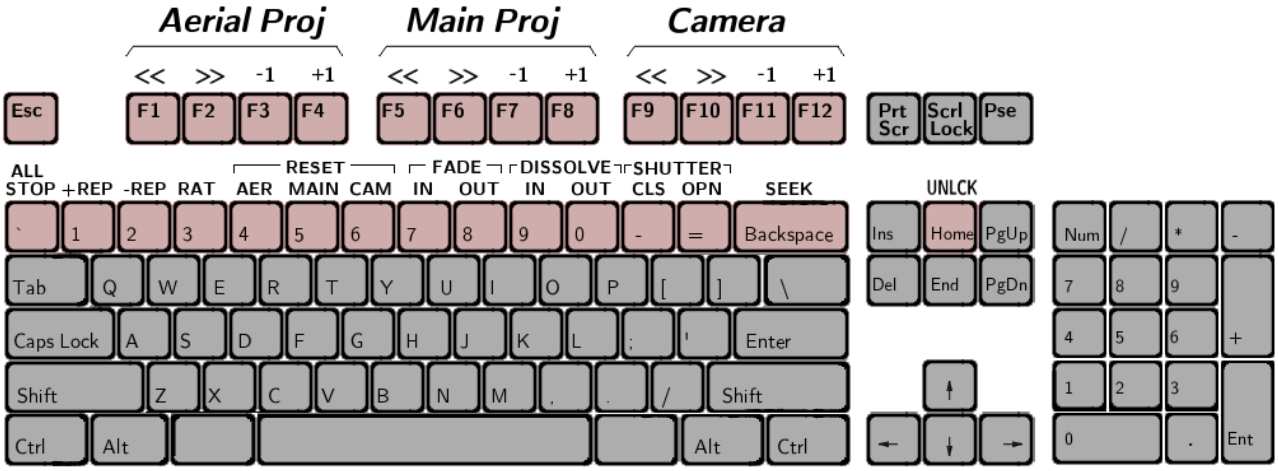
Often there are 'non-standard' utilities in the \bin directory. These can USUALLY be documented with the 'man' command. (e.g. HOME.EXE is documented with 'man home') These utilities are useful, and should be used by anyone willing to learn them.

*** SHOOTING **

PUSH BUTTON SHOOTING - THE 'KEY' COMMAND

The 'KEY' command allows the operator to use buttons to control the camera, projector and fader much like the mechanical printer controls that many people are used to.

The function keys across the top of the keyboard are windoff buttons. Number keys (below the function keys) control ratio shooting, counter resets, fade/dissolve setups, fader open/close, etc.



If you don't have a keyboard template which shows which keys do what, refer to the man pages on KEY(OPCS), and make your own. It is best to have some sort of template on the keyboard.

Hitting ESC or Q will exit the 'KEY' mode so you can execute any of the other OPCS commands.

The following shows common operations the cameraman wants to do, showing the keystrokes that do them. 'Operation' is the label on the keyboard template, 'Key to hit' is the actual keyboard key to hit.

Only in cases where "Then type:" is specified, it is assumed you should always hit ENTER after typing the specified values. [Space] is used to denote hitting the spacebar.

COUNTERS

To reset the camera counter to zero:

Operation	Key to hit	
RES CAM	**6**	Then type: **0 [Enter]**

For the main projector, same thing, but hit '5'. For the aerial projector, hit '4'.

To set the counters to other values, type the desired value.
To set the camera counter to 100:

Operation	Key to hit	
RES CAM	**6**	Then type: **100 [Enter]**

WINDOFF WITH SHUTTER CLOSED

The following shows how to cap the shutter, windoff some frames on the camera, then open the shutter again:

Operation	Key to hit	
CLS	**=**	

Then:

CAM FWD	**F9**	(Camera runs while F9 pressed)

Then:

OPN	**-**	

Windoff 100 frames on the camera, shutter automatically caps:

Operation	Key to hit	
SEEK	**BACKSPACE**	Then type: **100 [Enter]**

Windoff camera out to frame #304, shutter automatically caps:

Operation	Key to hit	
SEEK	**BACKSPACE**	Then type: **>304 [Enter]**

Send aerial to frame #200, main to #100, and windoff 50x on camera:

Operation	Key to hit	
SEEK	**BACKSPACE**	Then type: **>200 [Space] >100 [Space] 50 [Enter]**

STRAIGHT PRINTS

For a 1:1 step print with ONLY main projector (single projector system):

Operation	Key to hit	
RAT	**3**	Then type: **1 [Space] 1 [Enter]**

Then:

REP+	**1**	

..same thing, but on an aerial system:

Operation	Key to hit	
RAT	**3**	Then type: **0 [Space] 1 [Space] 1 [Enter]**

Then:

REP+	**1**	

For a 1:1:1 step print on an aerial system:

Operation	Key to hit	
RAT	**3**	Then type: **1 [Space] 1 [Space] 1 [Enter]**
Then:		
REP+	**1**	

STEP PRINTS

For a 2:1 step print:

Operation	Key to hit	
RAT	**3**	Then type: **2 [Space] 1 [Enter]**
Then:		
REP+	**1**	

For a 2:2:1 step print:

RAT	**3**	Then type: **2 [Space] 2 [Space] 1 [Enter]**
Then:		
REP+	**1**	

FADES/DISSOLVES

For a 24 frame fade in on a held projector frame:

Operation	Key to hit	
FDI	**7**	Then type: **24 [Enter]**
Then:		
CAMERA FWD	**F9**	

For a 24 frame fade in on a straight print, make sure your
ratio is currently 1:1 (See STRAIGHT PRINTS) then:

Operation	Key to hit
FDI	**7** Then type: **24 [Enter]**
Then:	
REP+	**1**

(REVISION 3.00 OR LATER)
To set up a fade in, hit the FADE IN button, and enter the number of frames for the fade. When you run the camera, the fade will take place. Fade outs and dissolves are set up the same way. You can cancel a fade by hitting the FADER OPEN or FADER CLOSE buttons.

NOVICE'S GUIDE TO DOS COMMANDS

Note that to run a DOS command, you should precede the command with a '!' when you are in the OPCS software. Or, if you prefer, you can set up more DOSCMD(OPCSDEFS) entries in your OPCSDEFS.OPC file so you can type in certain DOS commands without the need for using '!'.

*** Formatting Floppy Disks **

format a:

Be careful with the DOS 'format' command.. If you don't supply any arguments to format, it will often default to formatting the hard disk! An easy mistake to make.

*** Directory Listings ***

The DIR command is how to get directory listings:

```
dir                  # list files in the current directory
dir *.run            # list files that end in .run
dir m*.*             # list all files that start with 'm'
dir ilm/scn4a/*.*    # list all files in the ilm/scn4a directory
dir a:               # list files on floppy drive
```

Most versions of DOS support sorting of file listings.
Dos 6.0 supports these:

```
dir /on     # sort by name
dir /ox     # sort by extension
dir /od     # sort by date (oldest first)
dir /os     # sort by size
```

See '**dir /?**' for a list of all the options dir supports.

COPY FILES

The copy command has two arguments: a source and a destination. If the destination is not supplied, the current drive is assumed to be the destination.

```
copy *.run a:              # copy files ending in .run to floppy
copy a:*.run .             # copy files ending in .run FROM floppy
copy myfile.pos save.pos   # make a copy of myfile.pos to save.pos
```

DELETE FILES

When you delete files, they don't come back, so BE CAREFUL. Especially BE CAREFUL WHEN DOING WILDCARD DELETES!

```
del junk.pos      # deletes 'junk.pos' from disk
del a:crap.jnk    # deletes crap.jnk from floppy disk
del crap*.*       # deletes files that start with 'crap'
del *.*           # deletes ALL FILES - Watch out!
```

MAKING SUB DIRECTORIES

```
mkdir fred         # makes a subdir 'fred' in the current dir
mkdir fred\jobs    # makes a subdir 'jobs' in 'fred'
mkdir a:save       # makes a 'save' subdir on the floppy
```

CHANGING INTO/OUT OF SUBDIRECTORIES

```
cd fred            # go into 'fred' directory
cd jobs            # go into 'jobs' directory
cd ..              # go back one to 'fred'
cd                 # tells you what directory you are in
```

REMOVING SUB DIRECTORIES

```
del fred\job\*.*   # clean out JOB directory of all files first
rmdir fred\job     # removes JOB directory (FRED remains)
rmdir fred         # removes FRED directory
```

SEARCHING FOR FILES

The 'whereis' command will search the entire hard disk for filenames you supply as an argument, and shows the full pathname to any matching files. Examples:

```
whereis opcsdefs.opc  # searches for all OPCSDEFS.OPC files
whereis foo*.*        # searches for all files starting with "foo"
```

HARD DISK BACKUPS

You should definitely refer to your DOS manual for this one. See the docs on BACKUP and RESTORE commands. But here are a few examples:

HOW TO BACKUP THE C: DRIVE

```
mkdir a:\opcs
xcopy /s c:\opcs a:\opcs
```

HOW TO RESTORE THE C: DRIVE FROM BACKUPS

```
cd \
mkdir \opcs
xcopy /s a:\opcs c:\opcs a:
```

LOOKING AT TEXT FILES

```
type myfile.pos    # Type out myfile.pos in one blast
more myfile.pos    # View the file a page at a time
```

HOW TO EDIT THE HIDDEN C:\MSDOS.SYS FILE

The MSDOS.SYS file lets you disable Windows 95/Windows 98 from starting, so you can boot straight into DOS. But to edit the file, you have to turn off its hidden system attributes, then turn them back on when done:

```
attrib -S -H -R c:\msdos.sys    # Remove system/hidden/readonly
edit c:\msdos.sys               # Edit file, make changes
attrib +S +H +R c:\msdos.sys    # Restore system/hidden/readonly
```

The recommended contents of the MSDOS.SYS file to ensure only DOS boots (and not Windows) for OPCS is:

```
;FORMAT
[Options]
BootGUI=0
Logo=0
BootDelay=0
```

OPCS BOOT INSTALL

This text describes the boot process for OPCS, when the machine starts up. This assumes general familiarity with IBM PCs and DOS terminology.

It is assumed you have:

- An IBM PC with ISA or EISA slots
- An RTMC card or A800 card plugged into one of the ISA slots
- 512K or more of system memory.
- A hard disk with MS-DOS 6.xx, Win95, or Win98 installed
- For Win95/Win98: configured to boot into DOS mode (BootGUI=0)
- AUTOEXEC.BAT and CONFIG.SYS configured as described below

CONFIG.SYS

Your C:\CONFIG.SYS should at minimum have these (or similar) settings:

```
DEVICE=C:\WINDOWS\HIMEM.SYS
DEVICE=C:\WINDOWS\COMMAND\ANSI.SYS
DEVICE=C:\OPCS\BIN\OPCSBOLD.SYS
FILES=10
BUFFERS=20
```

HIMEM.SYS is optional but recommended.
ANSI.SYS is required but might be in a different directory.
Values for FILES/BUFFERS are minimum.
OPCSBOLD.SYS is required if 'nixie' counters are used.

AUTOEXEC.BAT

Your C:\AUTOEXEC.BAT should have at minimum these settings:

```
set PATH=\OPCS\BIN;%PATH%
set MANPATH=c:/OPCS/MAN/MAP
```

..and at least one of the following drivers should be started,
depending on which stepper motor pulse generator card you have
installed in the machine's ISA slot:

```
 A800DRV.COM  - for the a800 card
 RTMC48.COM   - for the RTMC48 or Kuper Industrial card
 MDRIVE.COM   - for the RTMC16 card
```

If started with no command line flags, the drivers assume
the cards are using the default jumper settings (usually
baseaddr=300, IRQ=5).

If your jumper settings are different than the defaults, be sure
to specify the appropriate command line options that match your
card's jumper settings.

Invoke the driver with the "-help" flag to see the driver's command
line options, e.g.

```
a800drv -help
rtmc48 -help
mdrive -help
```

If a driver doesn't show a help screen, that driver ONLY supports
the default jumper settings.

For example, if your A800 card has default settings, you can just put:

a800drv

..in your AUTOEXEC.BAT.

If, however, your A800 card has the BaseAddr set to 340, instead of the default 300, then you'd have to start the driver with:

```
a800drv -b340 -i5
           |    |
           |    IRQ5
           BaseAddr 340
```

ORIGIN

Gregory Ercolano, Topanga, California 04/12/00

This page intentionally left blank.

OPERATOR COMMANDS

```
INTRO(OPCS)             Optical Printer Control System          INTRO(OPCS)

OPCS MANUAL SECTIONS
    OPCS     - The operator commands reference and main application
    OPCSDEFS - The config file (OPCSDEFS.OPC) commands
    DOCS     - General docs on hardware, support tools, tutorials, etc.

INTRODUCTION
    If you're new to the OPCS software, start with:

       1) 'man quickref'     -- The operator's tutorial. See QUICKREF(DOCS)
       2) 'man -k OPCS:'     -- List all OPCS commands (see OPCS COMMANDS below)
       3) 'man -k OPCSDEFS:' -- List all config file commands
       4) 'man syntax'       -- General syntax of the OPCS commands
       5) 'man math'         -- How to use OPCS's built-in online calculator

    You can quickly list all the OPCS commands by running the '?' inside OPCS.
    For any commands you don't remember, just run 'man' followed by the command
    from that list to see the docs. (e.g. 'man cam', 'man pro', 'man fdi', etc.)

    What follows is a list of all OPCS commands with a one line description
    of each, which at the time of this writing is:

       !         - execute a system command (bang)
       #         - the # comment character (comment)
       autofilt  - enable/disable the auto-wedging filter wheel
       calc      - calculator usage/shortcuts
       cam       - shoot frames on just the camera
       check     - check counter values for specific channels
       chk       - check pro/cam/shu counters against arguments
       cls       - close the fader
       custom    - recommended custom commands implemented by local
       do        - repeat commands several times
       dxi       - set up a dissolve in
       dxo       - set up a dissolve out
       feed      - feed new posns to motors every frame
       fdi       - set up a fade in
       fdo       - set up a fade out
       go        - position motors go to new positions
       jog       - jog 'positioning' motors interactively
       key       - use keys to run motors
       load      - unseats projectors for loading film
       ldefs     - load a motor definitions file
       lineup    - (CUSTOM) seat camera for lineups
       load      - (CUSTOM) unseat projectors for film loading
       log       - log all entered commands to a file
       motors    - enable/disable motors for script debugging
       opn       - open the fader
       pro       - shoot frames only on the main projector
       pro1      - shoot frames only on the main projector (#1)
       pro2      - shoot frames only on the aerial projector (#2)
       pse       - will pause a running script
       rat       - change the current pro/cam ratio
       rep       - shoot the current pro/cam ratio
       run       - run an OPCS script file
       res       - reset/preset the projector/camera counters
       reset     - reset any channels to certain values
       seek      - seek to positions quickly on camera/projector(s)
       shu       - move the fader to a position
       show      - display current positions for all 8 motors
       spd       - change the camera's rotation speed
       unlock    - (CUSTOM) unlock motors for manual adjustment
       velrep    - load and run a .vrp file (velocity repeat)
```

```
For more details on how the OPCS application itself, see 'man opcs',
which describes OPCS briefly:

    > Command line editing hotkeys
    > Environment variables
    > Common error messages

SEE ALSO
        OPCS(DOCS) - OPCS main program
      SYNTAX(DOCS) - General syntax of OPCS commands
        A800(DOCS) - Notes on the A800 stepper motor control board
      RTMC48(DOCS) - Notes on the RTMC48 stepper motor control board
     CENTENT(DOCS) - Centent driver wiring
    QUICKREF(DOCS) - Camera operator quick reference       ***
    OPCSETUP(DOCS) - OPCS hardware/software setup details  ***
    OPCSHARD(DOCS) - OPCS hardware                         ***
     VERSION(DOCS) - OPCS version history

ORIGIN
    Gregory Ercolano, Los Feliz California 08/17/20
```

AUTOFILT(OPCS) Optical Printer Control System AUTOFILT(OPCS)

NAME
autofilt - enable/disable the auto-wedging filter wheel

USAGE
autofilt [-nohome] [on|off]

EXAMPLES
autofilt on # Enables auto-filter wheel movement,
sends the filter wheel to home position.

autofilt off # Disables auto filter wheel.
sends filter wheel to home position.

autofilt # Prints if autofilt mode is ON or OFF

autofilt -nohome on # Enables auto-filter wheel movement,
does NOT home or reset counter to zero.

DESCRIPTION
Sets up the filter wheel to move automatically each time the camera finishes shooting a frame. The filter wheel begins moving immediately after each exposure, while the shutter is closing.

To shoot a 20x wedge:

autofilt on cam 20 autofilt off

Both 'autofilt on' and 'autofilt off' automatically send the filter wheel to its home position (it invokes 'home autofilt') and zeroes the channel's counter. This behavior can be disabled with the **-nohome** option.

OPERATION
'autofilt on' will do the following:

> Executes 'home autofilt' (unless -nohome is specified) to home the wheel to a position that has NO FILTER.

> Zeroes the filter wheel channel's counter

> Like fades/dissolves, commands that involve exposing the camera (CAM, REP, etc) will first expose the filter in the gate BEFORE advancing the filter wheel. The filter will NOT move during SEEK(OPCS) or GO(OPCS).

SETUP NOTES
It is assumed the filter wheel speeds have been configured to move to the next filter position quickly enough so that it stops before the camera begins exposing the next frame. See FILTER(OPCSDEFS) for a full technical description.

SEE ALSO
FILTER(OPCSDEFS) - define channel to control a filter wheel

ORIGIN
Version K1.12e+ Gregory Ercolano, Venice California 04/10/98

```
!(OPCS)         Optical Printer Control System          !(OPCS)

    NAME
        ! - execute a DOS command

    USAGE
        ! [DOS command]
        - or -
        [OPCS cmds] ! [DOS cmd] ! [OPCS cmds] ! [DOS cmd]  ...

    EXAMPLES
        ! del /y junk.foo          # delete the file junk.foo
        ! dir *.run ! cam 5 pro 3  # run 'dir *.run', then shoot

    DESCRIPTION
        Execute a single command in MS-DOS without leaving the OPCS software.

        All characters to the right of the '!', up to the end of line,
        or '#', or another '!', are passed to DOS for execution.

        To allow several DOS commands on a single line, or DOS calls mixed
        with OPCS commands, you can make repeated use of the '!' character
        to switch back and forth within a single line:

            ! <DOS-command> ! <OPCS-command> ! <DOS-command> ..

        Example:

            ! copy *.pos a: ! cam 12 pro 12 ! dir a:

        The above runs 'copy *.pos a:' in DOS, then runs the CAM and PRO
        commands in OPCS, then runs the "dir a:" command in DOS again.

        Not only can you run DOS commands, but you can run DOS batch
        scripts (.BAT), perl scripts, or other C programs such as the
        HOME(DOCS) program.

    OTHER EXAMPLES
        !copy a:test .  # Copy the file 'fred' from the A: drive to
                        # the current directory.
        !command        # Run a DOS shell..'exit' returns to OPCS

    SEE ALSO
        DOSCMD(OPCSDEFS) - define DOS commands that don't need the '!' prefix

    ORIGIN
        Adapted after Unix utilities like vi(1) and ed(1).
```

CAM(OPCS) Optical Printer Control System CAM(OPCS)

NAME

cam - shoot frames on the camera

USAGE

```
cam [value]          # run camera so many frames fwd or reverse
```

EXAMPLES

```
cam 12               # run camera 12 frames forward
cam -12              # run camera 12 frames in reverse
cam 54'11            # run camera 54 feet 11 frames
cam >0               # run camera TO counter frame zero
cam >-32             # run camera TO counter frame -32
cam >34'8            # run camera TO counter 34 ft 8 frms
cam >(108/(10+2))    # run camera 9 frames (See SYNTAX(OPCS))
```

DESCRIPTION

This command will shoot the specified frames on the camera using the current shutter speed set by SPD(OPCS). The current shutter speed is always displayed at the top/right of the screen.

Negative numbers will shoot in reverse. So -100 will run the camera in reverse 100 frames.

Values can also be specified as feet/frames (e.g. 2'15), or as a mathematical expression (as shown above).

If the value is preceded by '>', this indicates the camera should "go to" the absolute camera frame counter value.

For fast windoffs that don't involve exposing film, use SEEK(OPCS) which uses the camera's slewing speed and caps the fader before moving the camera.

FADE/DX

If there are any pending fades or dissolves, the fader will FIRST move to its next position before the camera exposes a frame.

FEED

If FEED(OPCS) is enabled, the motors will /first/ move to position before the camera exposes a frame. (Similar to fade/dx)

AUTOFILT

If AUTOFILT(OPCS) is enabled, the camera will first expose the filter in the gate before moving the filter wheel. (Like the projector, what you see in the gate before shooting is what's about to be shot)

ALLSTOP

Hitting the ALLSTOP key (usually the (`) key) will halt the motors at the nearest frame. ALLSTOP is safe during shooting; it will not affect exposure, and always leaves the shutter closed after stopping. The allstop key can be redefined by ALLSTOP(OPCSDEFS).

BUCKLE/VIEWER

The camera WILL NOT RUN if the buckle is tripped. When exposing film, it also won't run if the viewer is open. If either condition occurs while the camera is shooting, the motors will stop with an error at the nearest frame, similar to ALLSTOP.

If 'hardware no' is in effect, the buckle/viewer state is ignored, so that the OPCS software can run on remote computers that are not physically connected to an optical printer.

TENSION MOTORS

Before the camera or projector move, the appropriate tension motors are enabled, to ensure take-ups are in correct modes for film motion.

COUNTER OVERFLOWS

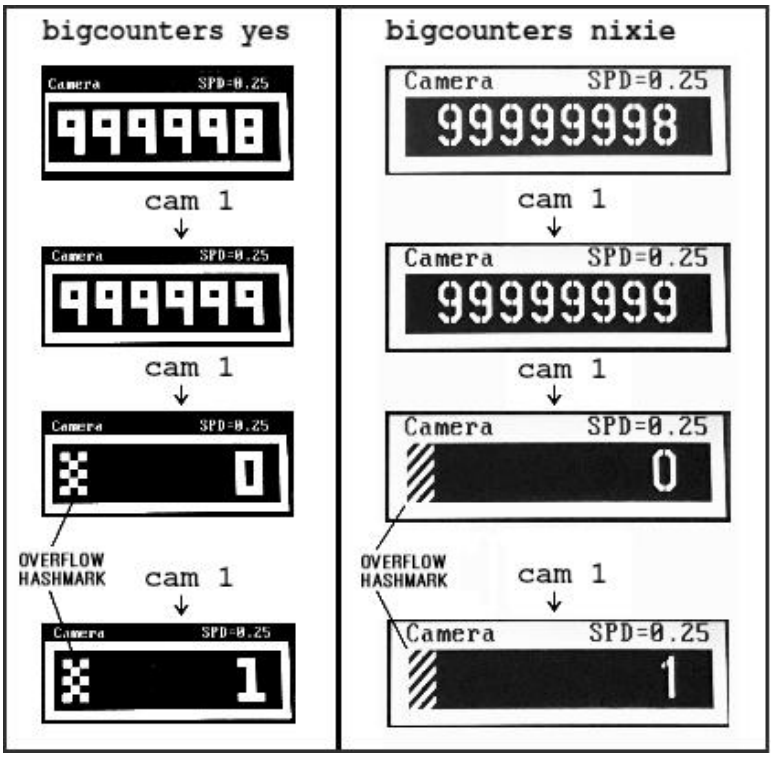
The software internally manages frame numbers in 32-bits, and therefore can handle values in the range of +/-2 billion.

However, the frame counter /display/ has a limited number of characters it can display, causing it to 'clock over' like the odometer of a car.

For 'bigcounters yes' (See BIGCOUNTERS(OPCSDEFS)), the limit is 6 digits, i.e. -99,999 thru 999,999.

For 'bigcounters nixie', in K2.10 and up supports 8 digits, i.e. -9,999,999 thru 99,999,999.

In these cases where the counter overflows, it 'clocks over' to zero. In version K2.10 and up, a hash flag appears at the left of the counter display, warning of counter overflow, e.g.:



..or in "ASCII art", that would be:

Similarly, negative underflows (counts below zero) clock to zero displaying a negative sign prefix.

For 'bigcounters yes', counter progression works this way, where '//' represents the hashmark:

Actual Frame	'bigcounter yes' Display	
-100,002	// -2	<-- wraps to -2, shows hashmark
-100,001	// -1	<-- wraps to -1, shows hashmark
-100,000	// 0	<-- wraps to -0, shows hashmark
-99,999	-99,999	
-98,999	-98,999	
:	:	
-1	-1	
0	0	
1	1	
:	:	
999,998	999,998	
999,999	999,999	
1,000,000	// 0	<-- wraps to 0, shows hashmark
1,000,001	// 1	<-- wraps to 1, shows hashmark
:	:	

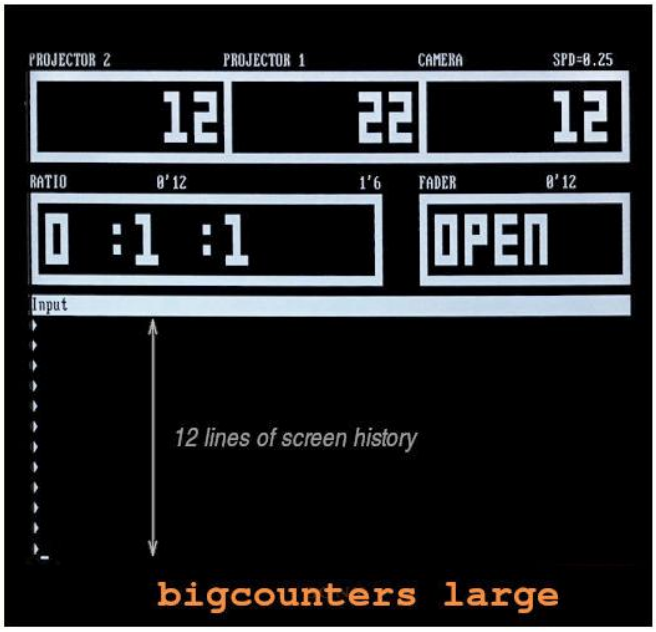
For 'bigcounters nixie', in version K2.10 and up, counter progression works this way:

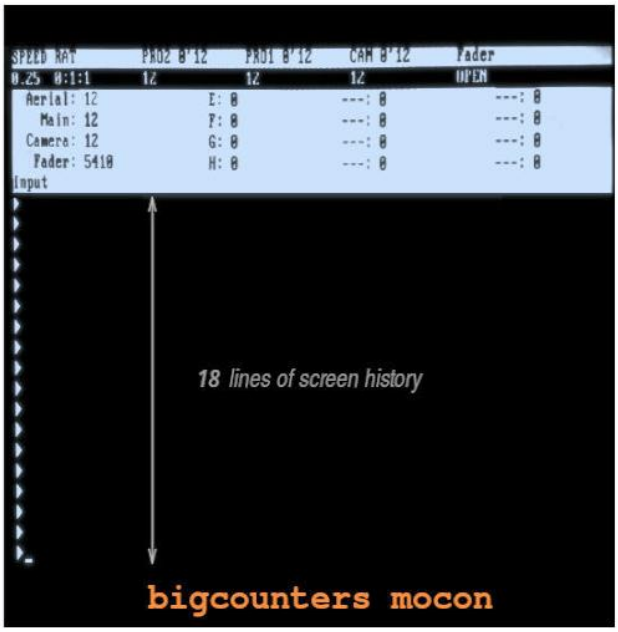
```
 Actual Frame  'bigcounters nixie' Display
 ------------  --------------------------
       :                 :
-10,000,002       //      -2    <-- wraps to -2, shows hashmark
-10,000,001       //      -1    <-- wraps to -1, shows hashmark
-10,000,000       //      -0    <-- wraps to -0, shows hashmark
  -9,999,999      -9,999,999
  -9,999,998      -9,999,998
       :                 :
      -1                -1
       0                 0
       1                 1
       :                 :
 99,999,998       99,999,998
 99,999,999       99,999,999
100,000,000       //       0    <-- wraps to 0, shows hashmark
100,000,001       //       1    <-- wraps to 1, shows hashmark
100,000,002       //       2    <-- wraps to 2, shows hashmark
       :                 :
```

This 'clock over' behavior is only true of the display; internally the software still keeps track of the /actual/ positions, so commands like 'cam >2000000' will still work correctly.

Note that 'bigcounters small' and 'bigcounters mocon' does not clip digits at all, and can display the full capabilities of 32bit values.

> Use '**bigcounters small**' to maximize operator's screen history
(21 lines of screen history)
> Use '**bigcounters mocon**' to monitor all channels for mocon moves.
(18 lines of screen history)
> Use '**bigcounters nixie**' for normal printing and medium sized counters.
(14 lines of screen history)
> Use '**bigcounters large**' for normal printing and largest counters.
(12 lines of screen history)





```
SEE ALSO
    -- Operator Commands (OPCS) --
    PRO / CAM  - run frames on the projector / camera
    RAT, REP   - tandem shooting (interlock)
    RES        - reset computer's pro/cam counters to new values
    CHK        - check if the pro/cam/shu counters are at certain values
    SEEK       - run the camera/projector at slewing speeds
    FEED       - feed motion control moves to motors every camera frame
    VELREP     - special purpose velocities for tandem shoots (eg. YCM)
    AUTOFILT   - enable/disable the auto-wedging filter wheel

    SHU        - send the fader shutter to absolute position in degrees
    OPN, CLS   - open/close the fader shutter
    DXI, DXO   - set up a dissolve
    FDI, FDO   - set up a fade

    MATH(DOCS)     - math expressions (for use in frame specifications)
    SYNTAX(OPCS)   - online calculator and OPCS math expression syntax
    MOTORS         - enable/disable motor hardware for debugging scripts

    -- Setup/Definitions (OPCSDEFS) --
    SPDINTERP      - configure exposure speed interpolations
    SPD            - configure camera's default exposure and slewing speeds
    RAMP           - configure camera's maximum accelerations and velocities
    MRP            - configure 'maximum ramp pulses' for shutter motors
    PPR            - configure 'pulses per revolution' for a motor

    BIGCOUNTERS    - enable/disable the large counter display
    FPF            - configure the 'frames-per-foot' (16mm, 35mm, Vista..)
    BUCKLE, VIEWER - configure buckle/viewer sensor's ports and bitmasks
    TENSION        - configure tension motors
    ALLSTOP        - define the ALLSTOP key
    HARDWARE       - enable/disable using printer hardware

HISTORY
    This command was in the first version of OPCS (Apple ][+).

ORIGIN
    Gregory Ercolano, Los Feliz California 12/18/89
```

```
CHECK(OPCS)            Optical Printer Control System          CHECK(OPCS)

    NAME
        check  - check counter values for specific channels

    USAGE
        check <chans> <val[,val..]>

        Where:
            <chans> is either a single channel letter, or a list of
                    channel letters with no spaces, e.g. "abcd"
            <val[,val..]> is either a single value, or a comma separated
                    list of counter values, one for every channel in <chans>

    EXAMPLES
        check f 1000        -- check f channel's counter is 1000
        check fg 1000,2000  -- check f counter is 1000 and g is 2000

    DESCRIPTION
        “check” verifies counter values are where they should be.
         Useful in RUN(OPCS) scripts to help catch errors during in script
         programming and shooting.

        If a check fails, an error warns of the discrepancy, prompting
         the operator with an ABORT/CONTINUE option.

        It's expected that for each of the <chans> specified, the exact same
        number of counter values are specified in the comma separated <val>
        list that follows.

        Counter values must be comma delimited, and can either be numeric,
        math expressions, or feet/frame format. Example:

            check ab (150-10),13'4   -- check if a's counter is 140
                                        and   if b's counter is 13'4

            check a (pro)            -- check if counter for aerial (a)
                                        has the same value as main (pro)

            check ab (cam+5),(cam+5) -- check if aerial (a) and main (b)
                                        counters are the same as cam's

        Note in OPCS the 'a' channel is the aerial projector, 'b' is the
        main projector, 'c' is camera, and 'd' is the fader shutter. Other
        channel assignments depend on your local configuration.

    HISTORY
        Early OPCS versions only managed cam/pro/shu, so CHK(OPCS) was designed
        for that. Motion control channels were later added, so the newer
        CHECK(OPCS) was added, allowing *any* motor channel to be checked.
        CHK(OPCS) is kept for backwards compatibility.

    SEE ALSO
        CHK(OPCS)    - older command that only checks frame counters and fader
        RUN(OPCS)    - run an OPCS script file
        MOTORS(OPCS) - enable/disable motors for debugging scripts

    ORIGIN
        Gregory Ercolano, Altadena California 05/28/20
```

CHK(OPCS) Optical Printer Control System CHK(OPCS)

NAME
chk - check if the pro/cam/fader counters are at certain values

USAGE
```
chk  [pro] [cam] [fader]           # single head printer
chk  [pro2] [pro1] [cam] [fader]   # dual headed printer
```

EXAMPLES
```
chk  10  13'12  0
     |     |    |
     |     |    Check if fader is closed
     |     Check if camera counter reads 13 feet 12 frames
     Check if projector counter reads '10'
```

DESCRIPTION
"chk" verifies counter values are where they should be.
Useful in RUN(OPCS) scripts to help catch errors during in script programming and shooting.

If a check fails, an error warns of the discrepancy, prompting the operator with an ABORT/CONTINUE option.

Arguments can be either in frames or in feet/frames format.
Arguments can also be selectively bypassed with '-', example:

```
chk - - 170       # check if fader at 170 degrees
chk 120 - - -     # check if pro2 counter at frame 120
```

EXAMPLE
Assuming the counters read: PRO=34, CAM=12, SHU=0, and the following command is executed from a running script:

```
chk 34 10 170
```

...all running scripts will stop, and the following message printed:

```
CHK:  Camera not at 10 frames
CHK:  Fader not at 170 degrees
Hit any key to continue  (ALLSTOP TO ABORT)
```

HISTORY
CHK(OPCS) is the old command limited to only checking cam/pro/fader.
CHECK(OPCS) is the newer command which can check **any** channels including motion control channels.

SEE ALSO
```
CHECK(OPCS)    - newer command that can check all counters
RUN(OPCS)      - run an OPCS script file
MOTORS(OPCS)   - enable/disable motors for debugging scripts
```

ORIGIN
Gregory Ercolano, Los Feliz California 11/29/89

COMMENT(OPCS) Optical Printer Control System COMMENT(OPCS)

NAME
'#' - the comment marker

USAGE
[comment text]

EXAMPLE
 # This is a valid comment
cam 12 pro 12 # This is also a valid comment

DESCRIPTION
The pound sign '#' is used as a delimiter for comment text. All text to the right of '#' up to the end of the line will be ignored by the OPCS command parser. This allows you to place text comments into run scripts without fear of having the comment executed as a command.

'#' can appear as the first character on a line, or after the last command on a line.

cam 12 pro 12 # Comment text can be anything

BUGS
None.

ORIGIN
Adapted after examples set in such UNIX utilities as SH, CSH, and many others. # is a standard way of delimiting comments under UNIX. UNIX is a trademark of AT&T.

CUSTOM(OPCS) Optical Printer Control System CUSTOM(OPCS)

NAME

custom - custom commands configured by the local site engineer

OVERVIEW

These are recommended custom commands, which are best implemented by the local site's engineer during setup, using RUNCMD(OPCSDEFS).

These are commands not built into OPCS, but rather added on during loading of the OPCSDEFS.OPC file, and assigned to scripts.

TYPICAL CUSTOM COMMANDS

---

load - Unseat projectors for loading
lineup - Seat camera for lineups
unlock - Unlock motors for manual adjustment
ycm - Y/C/M shooting

For more, see the scripts in the RUN directory, e.g. .\RUN\LOAD.RUN
For example, the 'ycm' custom command is implemented as three files:

ycm.run -- YCM 'runcmd' script
ycm.hlp -- YCM script's help file (describes command, args)
ycm.vrp -- YCM script's velrep file (velocities for YCM shooting)

There are example versions of these commands that come as part of the software, but you can define your own too. The RUNCMD definitions are in the opcsdefs.opc file (commented out), and the associated '.run' files are in the work\run* directory.

Simply edit the opcsdefs.opc file, and uncomment or create the commands you need, and modify the .run scripts to suit your needs.

Most of the 'example' commands come with man pages and/or .hlp files. Please refer to those for more information.

Custom commands that have man pages are:

LOAD(OPCS) - unseat projectors for film loading
LINEUP(OPCS) - seat the camera for lineups
UNLOCK(OPCS) - unlock motors for manual adjustment

SEE ALSO

RUNCMD(OPCSDEFS) - define your own OPCS command as a RUN script
DOSCMD(OPCSDEFS) - define DOS commands that don’t need the ‘!’ prefix
HOME(DOCS) - external command to home the motors

ORIGIN

Gregory Ercolano, Venice California 04/12/98

DO(OPCS) Optical Printer Control System DO(OPCS)

NAME
do - repeat commands

USAGE
do [count] [commands]
do until (expr) [commands]

EXAMPLES
do 12 rep 15 pro -5
^These commands repeat 12 times from left to right.

do until (cam>=45) rep 15 pro -5
^Loop until camera counter reaches 45 or greater.

DESCRIPTION
Repeats commands. Commands that follow the DO command up to the end of line are repeatedly executed the number of times specified by [count].

If 'DO UNTIL (expr)' is used, looping continues until (expr) is true.

DO commands can be nested within a line. That is to say, 2 DO commands can appear on the same line:

```
do 4  seek 10  do 8 rep 1 pro -1
          |        |
          |        These loop 32 times; the 'DO 4' loop
          |        executes the 'DO 8' loop 4 times.
          |
          These loop 4 times
```

In the case of **do until (expr)**, usually (expr) is a conditional expression that compares one of the counters to a value, e.g.:

```
do until (cam=12) ..  # until cam counter is 12
do until (cam>=12) .. # until cam counter greater than or equal to 12
do until (pro>200) .. # until main projector counter greater than 200
```

EXAMPLES
Run three 5 frame cycles of a moving projector image:

```
rat 1 1 do 3 rep 5 pro -5
             ^ These repeat 3 times from left to right.
```

Run twelve 4x cross dissolves on every 8th projector image, effectively 'weaving' still frames of a moving projector image:

```
do 12 dxo 4 cam 4 cam -4 pro 8 dxi 4  cam 4
      ^ These repeat 12 times
```

Run a script file 7 times:

```
do 7 run smallfile.run
     ^ This script is executed 7 times
```

Shoot a 12x wedge, allowing camera operator to manually load
ND filters into the projector's filter holder for each frame:

do 12 pse cam 1

ORIGIN
Gregory Ercolano, Los Feliz California 12/16/89

NAME

dxi/dxo - set up an automated dissolve in/out

USAGE

```
dxi [-x] frames    # set up a 'dissolve in'
dxo [-x] frames    # set up a 'dissolve out'
```

The **-x** option overrides the error message that warns you about doing a dissolve when the shutter isn't fully open/closed. This allows you to do dissolves from any fader position.

EXAMPLES

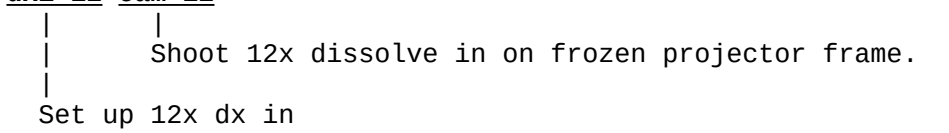
```
dxi 12       # Set up a 12 frame 'dissolve in'
dxi -x 12    # 12x dissolve, regardless of current fader position
```

DESCRIPTION

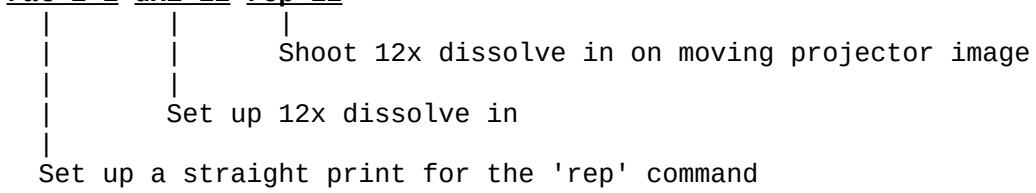
Sets up a dissolve. Once set, whenever the camera is told to expose frames (with CAM(OPCS), REP(OPCS), etc), the fader will automatically move to the proper position before exposing each frame.

Dissolves can operate on either still or moving images, depending on the command used for exposing film;

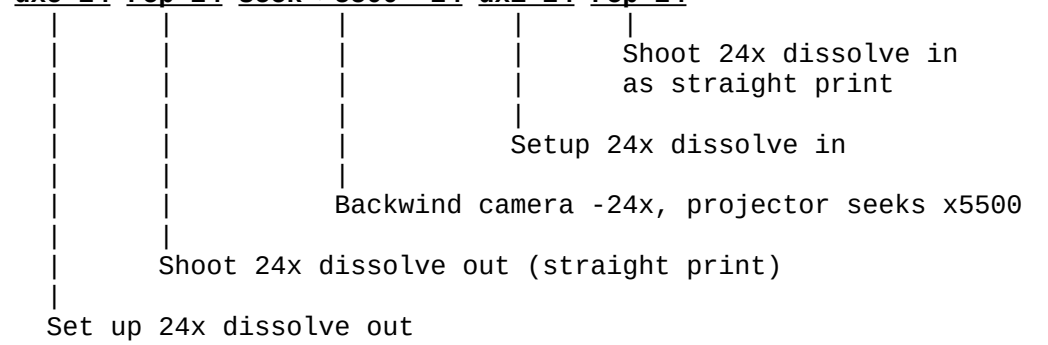
1) Setup and shoot a 12x "dissolve in" on a held projector image:



2) Setup and shoot 12x "dissolve in" as a straight print:



3) Setup and shoot a 24x 'lap dissolve' between two moving projector images. (Assumes '**rat 1 1**' in effect for straight print):



Commands can be 'all on one line' as shown above, or separate:

```
dxo 24            # Setup 24x dx out
rep 24            # Shoot 24x dx out as straight print
seek >5500 -24    # Backwind cam -24x, pro goes to frame #5500
dxi 24            # Setup 24x dx in
rep 24            # Shoot 24x dx in as straight print
```

Note: **seek >5500 -24** is effectively the same as **cam -24 pro >5500** though the latter may be slower to execute.

4) Shoot (12) 4 frame dissolves between every 8th projector image:

do 12 dxo 4 cam 4 seek -4 pro 8 dxi 4 cam 4
These commands repeat 12 times from left to right.

..which 'weaves' still frames from a slow-motion moving projector image.

NOTES

Dissolves remain in effect until all frames of the dissolve have shot. Any of the following commands will cancel an in-progress dissolve:

opn, **cls**, **shu**, **dxi 0**, **dxo 0**.

A status message near the fader's counter shows the remaining frames left for an in-progress dissolve.

If the -x flag is NOT supplied, the fader:

- o Must be fully OPEN before executing DXO
- o Must be fully CLOSED before executing DXI

..otherwise an error is shown. The only exception is if the **-x** flag is supplied, preventing these warnings, shooting the new dissolve based on the fader's *current* position.

Below are common errors:

dxi 12 cam 11 dxo 12 cam 12
^bombs here: only 11x into a 12x DXI.
opn dxi 12 cam 12
^bombs here: fader is already open

It's hard to see such errors in large '.run' scripts, so test with '**motors off**' to find such problems quickly.

DISSOLVE-SPECIFIC INSTALLATION NOTES

When setting up a new system, shoot extensive cross-dissolve tests to make sure the INTERP(OPCSDEFS) values for your fader are correct.

If interp values aren't accurate or there's mechanical slop in the shutter (see SLOP(OPCSDEFS)), you will see bumps in exposure. Test with different length lap-dissolves, and project on a large screen. Fine tuning SLOP and INTERP are essential for accurate dissolves.

SEE ALSO

OPCS Commands

CAM(OPCS) - shoot camera (fades/dissolves too)
OPN(OPCS), CLS(OPCS) - open/close fader shutter
SHU(OPCS) - move fader to an absolute position in degrees
DXI(OPCS), DXO(OPCS) - set up dissolve in/out
FDI(OPCS), FDO(OPCS) - set up fade in/out

OPCSDEFS Commands

FLOG(OPCSDEFS) - set Fader LOGarithmic curve for custom fades
FRANGE(OPCSDEFS) - set fade/dx's degrees range (for Hicon film stocks)
INTERP(OPCSDEFS) - set interpolation positions (fader, focus, etc)
SLOP(OPCSDEFS) - correct for slop in a motor (fader, focus, etc)

General

MATH(DOCS) - math expressions (for use in frame specifications)
SYNTAX(DOCS) - online calculator and OPCS math expression syntax

ORIGIN

Gregory Ercolano, Los Feliz California 11/29/89

NAME

fdi/fdo - set up an automated fade in/out

USAGE

```
fdi [-x] frames    # set up a 'fade in'
fdo [-x] frames    # set up a 'fade out'
```

The **-x** option overrides the error message that warns you about doing a fade when the shutter isn't fully open/closed. This allows you to do fades from any fader position.

EXAMPLES

```
fdi 24             # Set up a 24 frame fade in
fdi -x 24          # 24x fade, regardless of current fader position
```

DESCRIPTION

Sets up a fade. Once set, whenever the camera is told to expose frames (with CAM(OPCS), REP(OPCS), etc), the fader will automatically move to the proper position before exposing each frame.

Fades are based on a logarithmic curve that can be customized with FLOG(OPCSDEFS).

Fades can operate on either still or moving images, depending on the command used for exposing film;

```
1) fdi 12 cam 12          # fade in on held image
2) rat 1 1 fdi 12 rep 12  # fade in on moving image
3) rat 1 2 fdi 12 rep 6   # fade in with step print
```

#1 sets up and shoots a 12 frame fade-in on a still projector image.
#2 sets up and shoots a 12 frame fade-in on a moving projector image.
#3 sets up and shoots a 12 frame fade-out on a moving projector image on twos (step print).

Fades remain in effect until all frames of the fade have been shot. A fade can be canceled midway using any one of these commands:

opn, **cls**, **shu**, **fdi 0**, **fdo 0**.

The following effectively shoots the exact same thing:

```
> fdi 12 cam 12
> fdi 12 cam 6 cam 6
```

NOTES

> **fdi 0**, **fdo 0** or a SHU(OPCS) command with any value will cancel any pending fade or dissolve.

> A status message near the fader's counter shows how many frames are left to go for the fade.

> Normally, the fader must be:

- o OPEN before executing FDO
- o CLOSED before executing FDI

..otherwise an error is shown. The only exception is if the **-x** flag is supplied, preventing these warnings, shooting the new fade based on the fader's *current* position.

Below are some common errors:

```
fdi 12  cam 11 fdo 12  cam 12
               ^bombs here: only 11x into a 12x FDI.

opn  fdi 12  cam 12
     ^bombs here: fader is already open
```

It's hard to see such errors in large '.run' scripts, so it's advised to test scripts with '**motors off**' to find such problems quickly, without wasting film or waiting for motors to move.

SEE ALSO

OPCS Commands

```
CAM(OPCS)            - shoot camera (fades/dissolves too)
OPN(OPCS), CLS(OPCS) - open/close fader shutter
SHU(OPCS)            - move fader to an absolute position in degrees
DXI(OPCS), DXO(OPCS) - set up dissolve in/out
FDI(OPCS), FDO(OPCS) - set up fade in/out
```

OPCSDEFS Commands

```
FLOG(OPCSDEFS)   - set Fader LOGarithmic curve for custom fades
FRANGE(OPCSDEFS) - set fade/dx's degrees range (for Hicon film stocks)
INTERP(OPCSDEFS) - set interpolation positions (fader, focus, etc)
SLOP(OPCSDEFS)   - correct for slop in a motor (fader, focus, etc)
```

General

```
MATH(DOCS)       - math expressions (for use in frame specifications)
SYNTAX(DOCS)     - online calculator and OPCS math expression syntax
```

ORIGIN

Gregory Ercolano, Los Feliz California 11/29/89

FEED(OPCS) Optical Printer Control System FEED(OPCS)

NAME

feed - feed motion control moves to motors every camera frame

USAGE

```
feed [chans] [file]       # start feeding motors from file
feed off                  # disable feeding
```

[chans] declares specific channels to be used from the file, and should consist of combined channel letters (ie. 'efgh'), Asterisk (*) can be used to specify ALL channels.

[file] specifies the name of the file containing the positions for the channels.

[off] tells FEED to cancel any FEEDs in progress.

EXAMPLE

```
feed efgh zoomfile.pos    # Specify motion control file to
                          # control the e,f,g,h channels.

feed off                  # cancel any motion control files
                          # that may currently be in progress.

feed * zoomfile.pos       # all channels from file
                          # (note: abc chans are ignored)
```

DESCRIPTION

FEED sets up per-frame motion control for situations such as zooms and pans to be automatically controlled by an ASCII file containing columns of numbers that represent absolute positions.

When FEED has been given a file, any command that shoots a frame on the camera (such as **cam** or **rep**) will first read the next positions from the file, send the motors to their new positions, and then shoot the frame. This will continue until the end of the FEED file is reached, at which point the FEED file is automatically turned off.

The file will be read one line at a time for each frame exposed. When the end of the file is reached, the feed command is disabled, and the file is closed.

FEED files can be disabled prematurely by specifying **feed off**, canceling any FEED that is in effect, closing the file.

FEED is functionally similar to the FDI/FDO commands in that FEED does not actually shoot any frames, but sets things up so that when commands such as **cam** or **rep** are used, the motors move to the positions JUST BEFORE the camera shoots each frame.

POSITION FILES (xxx.pos)

Position files can be created in a text editor by typing in the values manually (see 'FEED FILE FORMAT' below), or by using external programs to generate the numbers (e.g. 'ease.exe' for generating ease-in/outs)

The file contains white space delimited columns of numbers. Each line in the file corresponds to a frame on the camera.

Each white-space delimited column corresponds to a motor axis or 'channel'; the first column being channel 'a', the second 'b', and so on.

Lines that start with '#' are ignored, so that comments can be included within the file for readability.

Example:

```
# zoom.pos file
# a     b     c     d     e     f     g     h     i ..
  0     0     0     0     0     0     505   100   0   <-- frame 1
  0     0     0     0   112     0     505   201   0   <-- frame 2
  0     0     0     0   340     0     505   302   0   <-- frame 3
  0     0     0     0   652     0     505   403   0   <-- frame 4
  0     0     0     0  1034     0     505   504   0   <-- frame 5
  0     0     0     0  1480     0     505   605   0   <-- frame 6
  0     0     0     0  1982     0     505   706   0   <-- frame 7
```

In this case the 'e', 'g' and 'h' channels have moves programmed into them; 'e' has a ramp, 'g' has a holding position of '505', and 'h' has a linear ramp. The rest of the channels are zero.

So to use just the 'e' channel from this file, the operator would use:

feed e zoom.pos rep 20

..this will set up and shoot the motion control move; each time the camera exposes a frame, the 'e' channel will move to the next position specified in the file under the 'e' column.

For more info on the position file format, see 'POSITION FILE FORMAT' below.

SPECIAL CHANNELS

Currently, FEED ignores channels ABC, even if specified. These correspond to the Aerial, Main, and Camera respectively, and are only controlled by shooting commands (PRO2, PRO, CAM, etc)

FEED is only for the positioning non-shutter motors.

Like the GO(OPCS) command, if you have zoom and follow focus channels configured on your system, FEED will do auto-follow focus ONLY if the focus channel is specified to FEED. The file need not contain any relevant values for the focus channel, but the channel must be specified in order to do a follow focus zoom.

Typically channels are assigned this way:

- a - aerial projector (if any)
- b - main projector
- c - camera
- d - fader (if any)
- e - zoom, usually the lens
- f - focus, usually the camera body
- g - filter wheel (if any)

Only the a/b/c/d channels must be assigned as shown; if there's a fader, it must be on the 'd' channel, the camera must be on 'c'.

The other channels (e/f/g/h..) can be assigned to any axis you like; the above are just recommendations.

FOLLOW FOCUS

In order for follow focus to work, the follow focus channel (f) must be specified to FEED. The values in the file for that channel will be ignored if INTERP is configured for followfocus, and can therefore be all zeroes.

The INTERP command will control the motor movement for this channel, automatically slaving the position of the focus to the positions on the zoom (e), based on the interpolations.

If the focus channel is not specified to FEED, zooms will move without moving the focus channel. In the following example, 'e' is the zoom, and 'f' is focus, with the 'f' channel slaved to the zoom with an INTERP command (not shown):

```
feed ef zoom.pos      # DO follow focus during zooms
feed e zoom.pos       # DON'T follow focus during zooms
```

If you want to specify your OWN focus positions in the FEED file, simply disable the INTERP(OPCSDEFS) command for the focus channel:

```
! echo interp f - 0 0 0 > foo.foo  ! ldefs foo.foo
```

This disables interpolation slaving for the focus channel, so that it can be run like a normal channel, using the values from the file for the positions.

POSITION FILE FORMAT

Position files are simple ascii files. They can be created with text editors, custom programs, or with software that comes with OPCS to create position files (e.g. ease.exe, gr.exe, etc).

- Lines whose first character starts with '#' are ignored. These are comment lines, and are not parsed by FEED(OPCS).
- Each line in the file is considered a 'frame'.
- Each line should have no more than 256 characters.
- Each number on the line represents an ABSOLUTE POSITION for that channel's motor. There is no way to represent RELATIVE positioning in a FEED file currently.
- The channels are always (ABCDEFGH..) respectively from left to right. Even though the software always ignores values for the 'abc' channels, some value (usually zeroes) must be there.
- Numbers are normally ASCII signed integers, although can be floating point values for channels such as the fader, to specify floating point 'degrees'.

Here is a sample file that has a 5 frame zoom programmed into the 'e' channel:

```
#  TEST.POS
# A  B  C  D  E   F  G  H
  0  0  0  0  10  0  0  0
  0  0  0  0  25  0  0  0
  0  0  0  0  45  0  0  0
  0  0  0  0  60  0  0  0
  0  0  0  0  65  0  0  0
```

Although values must be specified in the A,B and C columns, they are automatically ignored by the OPCS software to avoid screwing up the camera and projector positions which are better controlled by other commands such as CAM, PRO and REP.

To use the above file:

```
feed ef test.pos  # Start 'feed'ing numbers from test.pos
cam 5             # Shoot all 5 frames in the file
```

Note that you can use any shooting command, including RAT and REP to shoot the frames. The rule to remember is new positions are fed to the motors BEFORE the camera ever shoots a frame.

SEE ALSO

GO(OPCS) - Move motors some distance or to new positions
INTERP(OPCSDEFS) - configure channel position interpolations

ORIGIN

Gregory Ercolano, Los Feliz California 01/10/91

GO(OPCS) Optical Printer Control System GO(OPCS)

NAME
go - Move motors in steps or to absolute step positions

USAGE
go [chans] [distance[,distance,...]]
go [chans] [>position[,>position,...]]

EXAMPLES
go d -10 # channel d goes -10 steps from where it is
go efgh >0 # channels EFGH go TO position 0
go efg 120,130,>0 # e goes 120, f goes 130, and g goes TO 0
go ef 120 # ef go 120 steps from where they are

DESCRIPTION
Mostly for moving 'linear' motors such as zoom, follow-focus, etc.
Lets you move more than one channel at a time, each channel having
its own relative steps or absolute position, or all sharing one.

CAVEATS
1. When using GO to move shutter channels (cam, pro..), frame counters
will retain their frame counts, since GO moves steps, not frames.

2. Using GO on channels with interpolations are handled specially:

2a. When channels are slaved with INTERP, e.g. focus (f) slaved to zoom
(e), specify BOTH channels to GO so they move together, one slaved to
the other. Otherwise they will be moved separately. e.g.

go ef 12000 # 'e' moves 12000 steps, and slave 'f' moves
to maintain focus.

go e 12000 # 'e' moves 12000 steps. (f does not move)

go f 12000 # 'f' moves +12000 steps (e does not move)

go f >-30000 # 'f' moves to -30000 (e does not move)

2b. Alternatively, with a SINGLE CHANNEL interpolation (such as fader),
relative positions will move the channel in STEPS. Only when absolute
positions are specified (e.g. >170) will interpolations be used, e.g.:

go d 10 #'d' moves 10 STEPS. Relative positioning
of the fader always works in STEPS.

go d >170 #'d' moves to 170 DEGREES. Absolute positioning
of the fader always works in DEGREES.

SEE ALSO
SHOW(OPCS) - show current positions for all motors
JOG(OPCS) - interactively jog a positioning motor
INTERP(OPCSDEFS) - set interpolation positions (fader, focus, etc)
MATH(DOCS) - math expressions
SYNTAX(OPCS) - Online calculator and OPCS math expression syntax

ORIGIN
Gregory Ercolano, Los Feliz California 09/04/90

JOG(OPCS) Optical Printer Control System JOG(OPCS)

NAME
jog - jog motors interactively

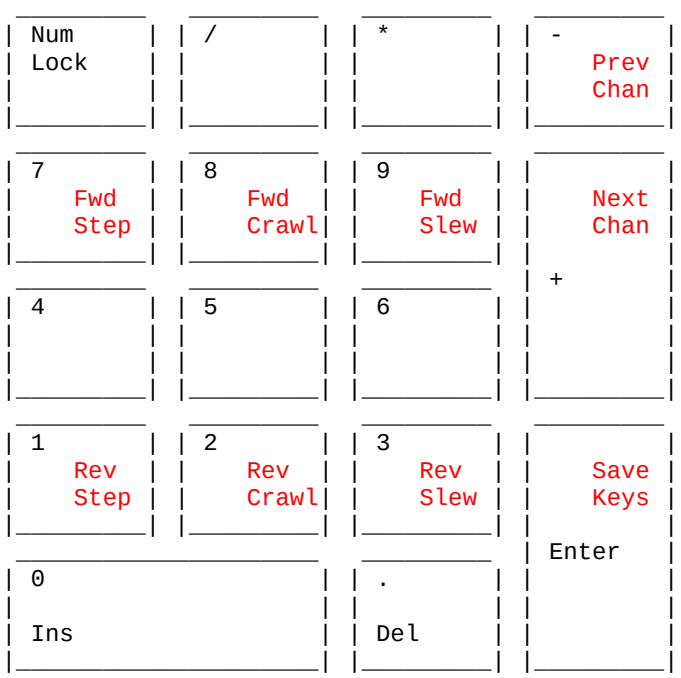
USAGE
```
jog                    # jog any channels
jog [channels]         # jog only specific channels
```

EXAMPLES

```
jog f          # jog the 'f' channel with the numeric keypad
jog            # jog ANY channels with numeric keypad
```

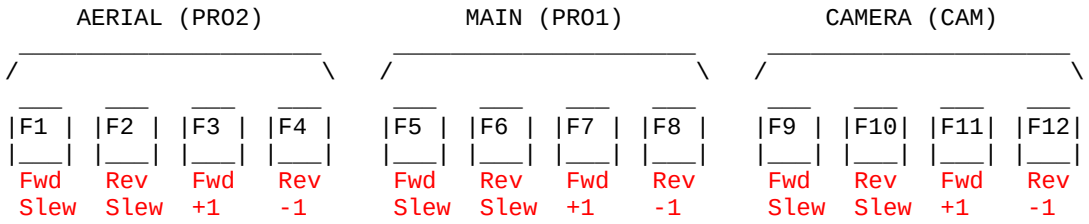
DESCRIPTION
This command brings up a menu oriented display where single key presses can run the printer.

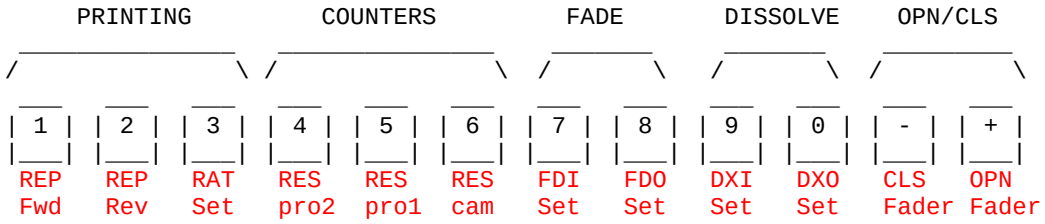
Keys on the numeric keypad are used for picking different motor channels and jogging the motors forward or reverse at 3 different speeds; Step, Crawl, and Slew:



You can redefine the number of pulses that STEP and CRAWL move per keypress. See JOGSTEP(OPCSDEFS).

Also, some of the KEY command keys are active as well. The function and number keys along the top of the keyboard allow the operator to run frames on the projectors and camera, as well as fade/dissolves, reset counters, rat, rep, etc.





Numeric Key Pad

Key	Description
1	Reverse in vernier mode. Hold to repeat.
2	Reverse in crawl mode. Hold to repeat.
3	Reverse in slew mode. Runs until key is released.
4	-
5	-
6	-
7	Forward in vernier mode. Hold to repeat.
8	Forward in crawl mode. Hold to repeat.
9	Forward in slew mode. Runs until key is released.
+	Select next channel to jog (i.e. 'a' becomes 'b')
-	Select previous channel to jog (i.e. 'b' becomes 'a')
ALT-A	Selects the 'a' channel
ALT-B	Selects the 'b' channel, etc..
V	Toggles 'Slave / No Slave' mode. [See COMMENTS]
Z	Sends current channel to '0' position
R	Reset counters [See COMMENTS]
U	Unlocks the motors (runs 'unlock' command)
ENTER	Saves current positions as a keyframe.
L	Load keyframe positions. [See COMMENTS]
E	Edit the keyframe list. See below 'KEYFRAME EDITOR'
S	Saves keyframes created with ENTER key, and E)ditor
C	Clears all keyframes so you can start again.

COMMENTS

V - If slaving is off, you can jog the zoom without moving the focus. With slaving on, moving zoom also moves focus.

R - In 'reset' mode, use Up/Dn arrows to select the channel to reset, and enter the new numeric value. Use ESC or Q to return to normal jog mode.

L - Loads the keyframe positions so they can be added to, edited, etc.

NOTES

If you are jogging positioning motors (not frame counter motors), and find that releasing the key does not immediately stop the motor, check the PPR(OPCSDEFS) setting for that channel. Normally positioning channels should have a low value like 10 for PPR.

When jogging the ZOOM, follow focus is not immediate when slewing. During a move, only the zoom will move until the key is released. At that point the focus will move to the appropriate focus position. This 'delay' is noticeable with especially with the SLEW keys (3&9).

GOTCHYAS

When using JOG(OPCS) inching keys on a channels like the camera and projector, the counters WILL NOT CHANGE, so as not to lose your frame counts for that channel.

BUGS

Currently the 0-9 keys (non-numeric keypad number keys) are not active during 'jog', although in the future they may echo the same functions the KEY command supports for those keys.

THE KEYFRAME EDITOR

The keyframe editor is a simple tool to help correct mistakes made while finding keyframes. It is a sub-menu of the JOG command, and is accessed as 'E' from the JOG menu.

UP/DOWN	Move the cursor to different keyframes. In the keyframe editor, the position list at the bottom right of the screen shows the positions FOR THE KEYFRAME the cursor is currently on.
PgUp/PgDn	If you have many keyframes saved, this will scroll through them a page at a time.
DELETE	Deletes the keyframe positions the cursor is currently sitting on, and all positions below it are 'moved up one' to replace the deleted keyframe.
INSERT	This inserts the current motor positions as a new keyframe where the cursor is sitting, and all the other key positions below it will be shifted down one.
G	Sends all motors to selected keyframe's positions.
ESC	Returns to the JOG mode.

COMMON KEYFRAME EDITING

DELETING ACCIDENTALLY SAVED KEYFRAME POSITIONS

If you save the same keyframe twice, you can correct the problem by entering the editor, find the extra position with the UP/DOWN cursor keys, and hit the DELETE key on the extra keyframe. Hit ESC to continue finding newer positions.

REPLACING BAD KEYFRAME POSITIONS WITH NEWER ONES

Assuming you saved a keyframe position, and later want to replace it with newly found positions, enter the editor with the motors at the newer positions. Find the old keyframe with the cursor and hit the DELETE key to remove it. Now hit the INSERT key to insert the current positions of the motors where the old positions were.

SEE ALSO

GO(OPCS)	- Move motors some distance or to new positions
KEY(OPCS)	- shoot camera/projector(s) frames by button control
KEYFUNC(OPCSDEFS)	- Lets you define which keys control motors
JOGSTEP(OPCSDEFS)	- set #steps for Pulse and Crawl modes

ORIGIN

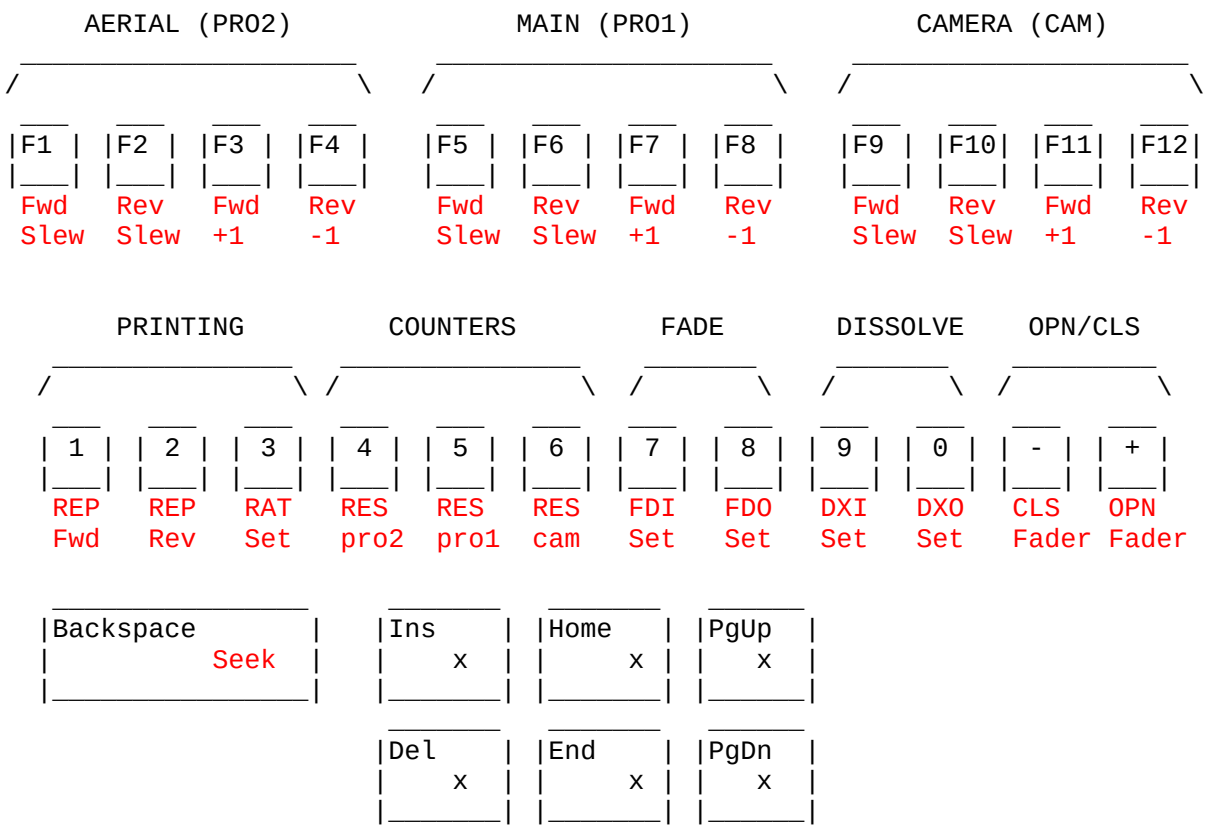
Gregory Ercolano, Los Feliz California 12/15/89

KEY(OPCS) Optical Printer Control System KEY(OPCS)

NAME
key - shoot camera/projector frames by button control

USAGE
key [no arguments]

DESCRIPTION
KEY(OPCS) gives the operator button controls to run camera, projector, and fader for simple shooting. These are the defaults defined by the KEYFUNC(OPCSDEFS) command in the OPCSDEFS.OPC file:



'x' indicates no default is configured, but can be customized using 'keydef' commands in the OPCSDEFS.OPC file

For a non-diagrammatic description:

```
                                        _
       F1 =  projector2 continuous run fwd  |
       F2 =  projector2 continuous run rev  |  AERIAL PROJECTOR SHOOT
       F3 =  projector2 fwd single frame    |         (PRO2)
       F4 =  projector2 rev single frame   _|
                                        _
       F5 =  projector1 continuous run fwd  |
       F6 =  projector1 continuous run rev  |  MAIN PROJECTOR SHOOT
       F7 =  projector1 fwd single frame    |         (PRO1)
       F8 =  projector1 rev single frame   _|
                                        _
       F9  =  camera continuous run fwd     |
       F10 =  camera continuous run rev     |  CAMERA SHOOT
       F11 =  camera fwd single frame       |
       F12 =  camera rev single frame      _|
                                        _
       1 =  run positive ratio shoot        |
       2 =  run negative ratio shoot        | RATIOS
       3 =  set ratio to new values        _|
                                        _
       4 =  set projector2 counter          |
       5 =  set projector1 counter          | COUNTERS
       6 =  set camera counter             _|
                                        _
       7 =  set fade in                     |  FADES
       8 =  set fade out                   _|
                                        _
       9 =  set dissolve in                 |  DISSOLVES
       0 =  set dissolve out               _|
                                        _
       - =  close fader                     |  OPEN/CLS
       = =  open fader                     _|
                                        _
BACKSPACE =  seek pro/cam                  _|  CAPPED SEEK
     ESC =  quit to OPCS command line
```

It is advised you add a keyboard template with the above values, or use a relegendable keypad like the "XKEYS Desktop PS/2 Keypad".

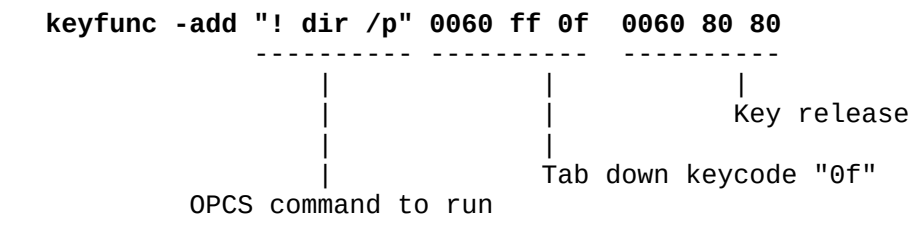
On most IBM keyboards, function keys (F1 - F12) are arranged across the top of the keyboard in 3 groups, 4 keys per group:

Axis	F-Key
Aerial Pro	F1 - F4
Main Pro	F5 - F8
Camera	F9 - F12

In each group, the first two keys shoot fwd/rev as long as you hold the key down. The others shoot single frame forward/reverse.

Use KEYFUNC(OPCSDEFS) to define external port bits so your own custom hardware buttons can control the motors. You can also tie your own custom commands and run scripts to buttons and keys.

EXAMPLE. To program the TAB key to invoke a DOS 'dir /p' command, put the following in your OPCSDEFS.OPC file:



Then restart opcs and run the 'key' command. Hitting TAB will show a directory listing, then you will be prompted to hit a key, then you're returned to the key mode.

SEE ALSO

JOG(OPCS)	- jog to new positions (in pulses) interactively
QUICKREF(DOCS)	- See PUSH BUTTON SHOOTING for examples
KEYFUNC(OPCSDEFS)	- Lets you define which keys control motors
MOTORS(OPCS)	- Disable motor hardware, simulates shutter runs

ORIGIN

Gregory Ercolano, Los Feliz California 02/15/91

LDEFS(OPCS) Optical Printer Control System LDEFS(OPCS)

NAME
ldefs - load OPCS definitions file

USAGE
ldefs filename.opc
ldefs -c <command and args>

EXAMPLES
ldefs hicon.opc # load OPCSDEFS cmds from hicon.opc
ldefs -c bigcounters nixie # use nixie counters
ldefs -c ramp a 10 180 10 150 # redefine ramps for a chan

DESCRIPTION
This command allows the operator to load other definitions files. Users can make copies of the OPCSDEFS.OPC file, and make changes to the copy, then load this new copy with the LDEFS command to put the changes into effect. This avoids modification of the original OPCSDEFS.OPC file.

One can also run single line OPCSDEFS commands with the '-c' flag, which interprets all arguments to the end of the line or a '!' character (see BANG(OPCS)) as OPCSDEFS commands. Example:

ldefs -c bigcounters nixie
ldefs -c name a Aerial name b Main name c Cam ! echo OK

OPCSDEFS files contain special commands that setup the OPCS system's internal parameters. Use 'man -k OPCSDEFS:' for a listing of all the OPCSDEFS commands (such as the 'opcsdefs.opc' loaded on startup), or for any other files/commands used with LDEFS(OPCS).

HISTORY
The '-c' flag was added in OPCS version K2.00 to allow immediate execution of defs commands. In older releases to do the same, you had to first write commands into a temp file (e.g., using ECHO), then load that, e.g.:

! echo bigcounters off > tmpfile
ldefs tmpfile ! del tmpfile

This trick is no longer needed in K2.xx, as you can use just '**ldefs -c bigcounters off**' for the same effect.

TRICKS WITH DEFS FILES
People familiar with the IBM's operating system will be familiar with these capabilities...

First, note that in K2.00 (and up), 'ldefs -c' can be used to run OPCSDEFS commands inside OPCS, e.g.:

ldefs -c bigcounters on # big counters

Which makes many of the below techniques unnecessary extra work. However, in the older releases (K1.xx) these are unavailable, so the below techniques must be used.

As with all DEFS file commands, you can execute motor definition commands from within the OPCS software by creating a small file, and the loading commands from it via LDEFS(OPCS)... In the following example, we switch back and forth between large and small counters:

```
! echo bigcounters on  > tmpfile ! ldefs tmpfile   # big counters
! echo bigcounters off > tmpfile ! ldefs tmpfile   # small counters
```

This 'trick' can be used with any OPCSDEFS commands, and uses the operating system's ECHO command and 'reroute output' symbol (>) to create the file FOO, which is then loaded as a file with the LDEFS command. This technique CAN be used within a script or when entering commands manually.

You can create multiline files from within a script as shown in this example using MSDOS's > and >> (append) symbols:

```
! echo flog 2.0         > tmpfile
! echo logcounters yes >> tmpfile
ldefs  tmpfile
```

This technique can be programmed into run scripts, so defs file information can be changed on the fly.

Here is another way to enter DEFS commands directly to the LDEFS command from within the OPCS software:

```
ldefs con            # Load the special MSDOS file CON...
logcounters no       # which is really the keyboard (console)
ppr a 400            # reading these commands from keyboard
^Z                   # CTRL-Z and RETURN ends this mode..
cam 12               # ..back to OPCS commands
```

The '**ldefs con**' technique works well for interactive typing, but cannot be programmed into a script, since it always reads from the keyboard. Use the 'echo' technique listed in the previous example for programming DEFS commands into a running script.

These techniques are actually standard ways of using the DOS operating system, and are not particular to just the OPCS software. They can be used by any program running under MSDOS that properly supports the operating system.

Users not familiar with these techniques should learn them only if they think they might need them. At very least, operators should be aware of these capabilities.

SEE ALSO

```
ECHO(OPCSDEFS)   - disable echoing of defs commands
OPCSCMD(OPCS)    - run OPCS commands from within OPCSDEFS command mode
man -k OPCSDEFS: - list OPCSDEFS commands with one line descriptions
man -k OPCS:     - list OPCS commands with one line descriptions
```

ORIGIN

Gregory Ercolano, Los Feliz California 11/29/89

l

LINEUP(OPCS) Optical Printer Control System LINEUP(OPCS)

NAME

lineup - (CUSTOM) seat the camera for lineups

USAGE

lineup [no arguments]

DESCRIPTION

Seats the camera so a lineup can be done.

This command is actually a script defined with a RUNCMD(OPCSDEFS) command, and is normally customized by your local site engineer.

This command normally does the following operations:

> Seat the camera

> Wait for user to hit a key

> Unseat the camera

INSTALLATION NOTES

An example implementation of the lineup command might be done as follows. Add the following command to the 'runcmd' section of your opcsdefs.opc file:

runcmd lineup lineup.run 0

..then create a file called 'lineup.run' which contains the following text:

```
@ # Seat camera. 'go' won't affect camera counters.
@ go c 1000
##### CAMERA SEATED FOR LINEUP ###
@ pse -noabort
##### CAMERA BACK TO NORMAL ###
@ # Unseat camera by moving back 1000 pulses.
@ go c -1000
```

A slightly more colorful version, assuming you know how to enter ANSI and control characters into your text editor:

```
@ go c 1000
#<BS><ESC>[1m*** <ESC>[5mCAMERA SEATED FOR LINEUP<ESC>[0m<ESC>[1m ***<ESC>[0m
@ pse -noabort
#<BS><ESC>[K<ESC>[A<ESC>[A<ESC>[K
@ go c -1000
```

ORIGIN

Gregory Ercolano, Los Feliz California 10/12/90

LOAD(OPCS) Optical Printer Control System LOAD(OPCS)

NAME
load - (CUSTOM) unseat projectors for film loading

USAGE
load [no arguments]

DESCRIPTION
Unseats the projector shuttles for easy loading.

This command is actually a script defined with a RUNCMD(OPCSDEFS) command, and is normally customized by your local site engineer.

This command normally does the following operations:

> Disables the tension motors for the camera and projector

> Move projector(s) to unseated position with GO(OPCS)

> Pause to allow user to load the film

> Return projector(s) to seated position with GO(OPCS)

> Enable the tension motors again

INSTALLATION NOTES
An example implementation of the load command might be done as follows. Add the following command to the 'runcmd' section of your opcsdefs.opc file:

runcmd load load.run 0

..then create a file called 'load.run' which contains the following text:

```
@ # Deenergize tension motors. Clearing port 379 bits 0 & 1.
@ ! echo @ clrbit 0379 03 00 > foo.defs ! ldefs foo.defs
@ go ab 1000,-1000
# *** UNSEATED FOR LOADING ***
@ pse -noabort
@ go ab -1000,1000
@ # Energize tension motors by setting the bits.
@ ! echo @ setbit 0379 03 00 > foo.defs ! ldefs foo.defs
```

Note use of leading '@' signs to disable echoing of the commands, to avoid cluttering the screen with unwanted text.

ANSI characters can be added to the script to embolden messages, and erase them when the user hits a key to continue.

ORIGIN
Gregory Ercolano, Los Feliz California 12/15/89

LOG(OPCS) Optical Printer Control System LOG(OPCS)

NAME
log - logs manually entered commands to a file or device

USAGE
log [-r] <filename.log>
log -f <filename> <format string>

EXAMPLES
log myfile # Start logging commands to MYFILE.LOG
log off # Turn off logging.
log -f tmp.log Camera Feet=%1cF Camera Frames=%1cp%n
Append a message to 'tmp.log'
log <MM-DD-YY> # creates date-stamped log filename

DESCRIPTION
The 'log <file>' and 'log off' usage of this command will enable/disable logging all commands you enter from the keyboard to a file. (Commands in RUN(OPCS) scripts will not be logged.)

The 'log -f <file> <message>' usage of this command will append a message to <file>, where <message> may contain formatting characters (see LOGFORMAT(OPCSDEFS)) that lets you embed counter values in the message.

If <filename> is the string "<MM-DD-YY>", then a date-stamped filename will be created. Example: if the date is Dec 31 2007 and you run "**log <MM-DD-YY>**", the resulting filename will be:

logs/12-31-07.log

If the .\logs directory does not exist, it is created.

'**LOG OUTPUT.LOG**' AND '**LOG OFF**'
When you quit the software, or enter LOG OFF, the log will be closed, and can later be viewed for reference.

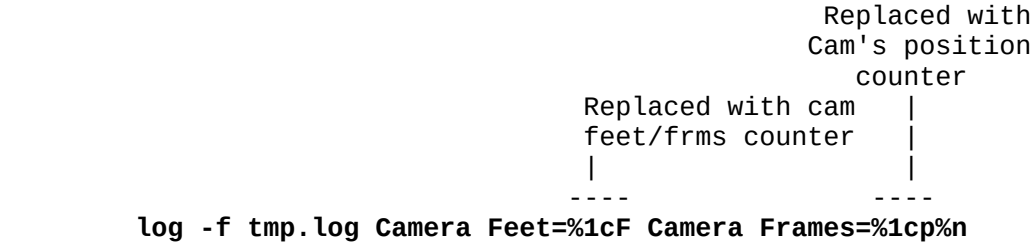
Optionally, the counters can also be logged to the file. See the LOGCOUNTERS(OPCSDEFS) man page for more on this.

To enable logging, specify some filename as an argument. You may also use LPT3:, COM1:, or other DOS device names to log directly to line printers, etc.

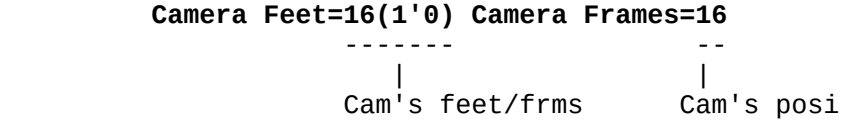
-r can be specified before the filename to include commands in executing RUN(OPCS) scripts to also be logged to the file. If -r isn't specified, only commands typed at the keyboard will be logged. (-r in OPCS K1.13b+)

To disable logging, type **log off**.

'LOG -F <FILENAME> <MESSAGE>'
With the -f flag, you can log a single message to a file with embedded counter values. For instance, if you want to append a message to a log file that includes the current camera counter:



See LOGFMT(OPCSDEFS) for a list of all the '%' format codes. The above example would append the following text to 'tmp.log', based on what the camera counter reads at the time:



NOTES

If a file already exists when you start logging to it, messages will be *appended* to the existing file. If you want to start with a fresh log file, remove it before logging to it, e.g.:

! del mylog.log ! log myfile.log

The LOGCOUNTERS(OPCSDEFS) command controls whether the log records the current counter positions or not. If you do not want counter data in your logs, add this to your OPCSDEFS.OPC file:

logcounters off

..or from within OPCS you can use:

! echo logcounters off > foo ! ldefs foo

Counter information lines are always preceded by a '#' comment character so the log file can be run as a script with RUN(OPCS), without the counter data being executed as OPCS commands.

It is advisable to use '.LOG' as the extension for log files so you can differentiate them from other OPCS files.

CARRIAGE RETURNS IN '**log -f**'

You must include a '%n' at the end of your <message> for the message to have a CRLF at the end. Otherwise, the line will remain unterminated, letting you concatenate to a single line using separate LOG(OPCS) commands, e.g.:

```
log -f tmp.log Camera=%1cF,
log -f tmp.log Projector=%1bF%n
```

..results in appending the following single line to 'tmp.log'

```
Camera=16(1'0),Projector=0(0'0)
```

To have those appear on separate lines, make sure both commands include a %n on the end:

```
log -f tmp.log Camera=%1cF%n
log -f tmp.log Projector=%1bF%n
```

..which results in the following two lines appended to 'tmp.log':

```
Camera=16(1'0)
Projector=0(0'0)
```

LINE PRINTERS

You can use LOG(OPCS) to maintain a continuous hardcopy printout as the user enters commands. EXAMPLE:

```
log lpt3            # log commands to the LPT3 line printer
```

Keep in mind that some of the parallel ports may be being used to control motor hardware.

CAVEATS/WARNINGS

OPCS log files should not be edited by word processors that introduce non-ASCII characters. Use 'edit' or 'vi' which are pre-installed.

Don't try to edit a log file while it's still logging, or you'll get unexpected results. Be sure to run '**log off**' before editing.

BUGS

If the operator uses ALLSTOP during command logging, it would be unwise to later execute the log file as a run script without making the proper modifications to the commands that were interrupted. Here is a sample log with a command that was interrupted:

```
#  1:1   0(0'0)         20(1'4)       CLOSED 0
cam -120

# ### OPERATOR HIT ALLSTOP KEY
#  1:1   0(0'0)         18(1'2)       CLOSED 0
```

Note the camera counter now reads 18 instead of -100. Because the command was interrupted, it never got to finish shooting. This could cause confusion later if this log were executed as a RUN script, and commands that followed used absolute positioning (cam >134). The command **cam -120** should then modified by hand:

cam >18 or **cam -2**

...to reflect the command as it was actually executed.

SEE ALSO

LOG(OPCS)	- log all commands entered by the user
RUN(OPCS)	- run a log file
LOGCOUNTERS(OPCSDEFS)	- enable/disable logging counters to logfiles
LOGFORMAT(OPCSDEFS)	- formats how values are printed to logfile

ORIGIN

Gregory Ercolano, Los Feliz California 11/29/89

MOTORS(OPCS) Optical Printer Control System MOTORS(OPCS)

NAME
motors - enable/disable the motor hardware for debugging scripts

USAGE
motors [on|off]

DESCRIPTION
With off as an argument, this command will disable motor movement, causing any commands that run motors (pro,cam,rep,go..) to NOT move the motors. Counters will run, but the motors will not. Also buckle/viewer checks will be disabled, not checking the actual hardware.

This command is useful for testing OPCS without motors and without any actual hardware connected, such as developing camera 'run' scripts on a portable computer, to be run later on an actual printer.

RUN(OPCS) scripts will execute faster when 'motors off' is in effect, allowing one to rapidly debug complex scripts.

When motors are disabled with this command, the current position of the motors are saved. When you re-enable the motors with 'motors on', the counters will revert to the previous positions just before 'motors off' was issued, avoiding confusion over where the motors REALLY are.

ENVIRONMENT

OPCS_NOMOTOR_FRAME_DELAY
(NEW IN K2.21/TC) This environment variable can be set before OPCS is started to configure a per-frame msec delay for 'motors off' simulated shutter runs for camera/projectors.

If unset or set to zero, 'motors off' shutter runs are instantaneous, the default behavior, enabling rapid debugging of RUN(OPCS) scripts.

However, it is sometimes useful for simulated shutter runs to consume at least some per-frame time. Setting this variable to e.g. 250 msecs causes a 1/4 second delay per simulated frame.

This variable can be set in AUTOEXEC.BAT, or anytime in DOS before OPCS is started. For example, this sets the delay to 250ms (1/4 sec):

```
set OPCS_NOMOTOR_FRAME_DELAY=250
```

..and this disable the delay (same as if the variable is unset):

```
set OPCS_NOMOTOR_FRAME_DELAY=0
```

EXAMPLES

```
cam 12        # Runs the camera 12x
motors off    # Disable running the motors
cam 12        # Counter will advance 12x, but motor wont run
motors on     # Enable motors. Counters revert to actual posn.
cam 12        # Run the camera normally
```

SEE ALSO
CHK(OPCS) - check if counters are where they're supposed to be
RUN(OPCS) - run OPCS command scripts (quickly with 'motors off')

OPN/CLS(OPCS) Optical Printer Control System OPN/CLS(OPCS)

NAME

opn - open the fader shutter
cls - close the fader shutter

EXAMPLES

opn # opens the fader
cls # closes the fader

DESCRIPTION

opn and cls allow the user to open or close the fader directly.

To move the fader to positions other than OPEN or CLOSED, use the SHU(OPCS) command to specify absolute degree positions for the fader.

NOTES

Specifying OPN or CLS during a pending fade or dissolve will effectively cancel the fade/dx, and send the fader to the specified position.

The system will not acknowledge the ALLSTOP key until AFTER the fader has completed running to its position.

OPN uses the INTERP(OPCSDEFS) command's [high] value to determine the degree position for OPEN on your system, and the [low] value to determine the degree position for CLOSED. Most cameras have 170 degree shutters, but some have 120. See INTERP(OPCSDEFS) for details.

SEE ALSO

OPCS Commands

CAM(OPCS) - shoot camera (fades/dissolves too)
OPN(OPCS), CLS(OPCS) - open/close fader shutter
SHU(OPCS) - move fader to an absolute position in degrees
DXI(OPCS), DXO(OPCS) - set up dissolve in/out
FDI(OPCS), FDO(OPCS) - set up fade in/out

OPCSDEFS Commands

FLOG(OPCSDEFS) - set Fader LOGarithmic curve for custom fades
FRANGE(OPCSDEFS) - set fade/dx's degrees range (for Hicon film stocks)
INTERP(OPCSDEFS) - set interpolation positions (fader, focus, etc)
SLOP(OPCSDEFS) - correct for slop in a motor (fader, focus, etc)

General

MATH(DOCS) - math expressions (for use in frame specifications)
SYNTAX(DOCS) - online calculator and OPCS math expression syntax
INTERP(OPCSDEFS) - setup interpolations for fader, etc.

ORIGIN

Gregory Ercolano, Los Feliz California 11/29/89

```
PRO(OPCS)              Optical Printer Control System          PRO(OPCS)

    NAME
        pro  - run the projector so many frames forward or reverse
        pro2 - for two-headed printers, runs projector #2

    USAGE
        pro  [frame spec]   # run main projector fwd or reverse
        pro2 [frame spec]   # run aerial projector (projector2) fwd or reverse

    EXAMPLES
        pro 12            # run projector 12 frames forward
        pro -12           # run proj 12 frames in reverse
        pro 54'11         # run proj 54 feet 11 frames
        pro >0            # run proj TO counter frame zero
        pro >1200         # run proj TO counter frame 1200
        pro >-32          # run proj TO counter frame '-32'
        pro >34'8         # run proj TO counter '34 feet 8 frames'
        pro (sqrt(9)+1)   # run proj 4 frames (3+1)

   DESCRIPTION
        This command will shoot the specified frames on the projector.
        Negative numbers will shoot in reverse. So -100 will run the
        projector in reverse 100 frames.

        Values can also be specified as feet/frames (e.g. 2'15), or as
        a mathematical expression (as shown above).

        If the value is preceded by '>', this indicates the projector
        should "go to" the absolute projector frame counter value.

        For dual headed printers, the PRO2 command will run the second
        projector. For compatibility there is a PRO1 command, which is
        the same as the PRO command.

    IMPORTANT NOTES
        OPCS always runs the projector at its fastest speed. See the
        SPD(OPCSDEFS) command for setting the projector's fast speed.

        Running the projector alone will have no effect on fades or dissolves,
        and the viewer may be open during projector runs without causing
        'VIEWER OPEN' errors, allowing viewing scenes in motion.

    ALLSTOP
        Hitting the ALLSTOP key (usually the (`) key) will halt the motors
        at the nearest frame. ALLSTOP is safe during shooting; it will not
        affect exposure, and always leaves the shutter closed after stopping.
        The allstop key can be redefined by ALLSTOP(OPCSDEFS).

    COUNTER OVERFLOWS
        For more info on how counter overflows are handled, see CAM(OPCS).

    BUGS
        See COUNTER note above.
```

```
    SEE ALSO
        PRO(OPCS)        - run frames on the projector (pro2 also)
        CAM(OPCS)        - run frames on the camera
        SEEK(OPCS)       - run the camera/projector at slewing speeds
        AUTOFILT(OPCS)   - enable/disable the auto-wedging filter wheel
        FEED(OPCS)       - feed motion control moves to motors every camera frame
        MATH(DOCS)       - math expressions (for use in frame specifications)
        SYNTAX(OPCS)     - Online calculator and OPCS math expression syntax
        VELREP(OPCSDEFS) - special purpose velocities for tandem shoots (e.g.. YCM
shooting)

    ORIGIN
        Gregory Ercolano, Los Feliz California 12/18/89
```

PSE(OPCS) Optical Printer Control System PSE(OPCS)

NAME
pse - pause a run script

USAGE
pse [-noabort] [-nodeb]

DESCRIPTION
This command will pause a running script to allow users to adjust filters, f-stops, and the like. This command simply prompts the operator with a '**Hit RETURN to continue**' prompt.

If the **-noabort** argument is specified, the user will not be able to hit 'allstop' to abort the PSE command, which can be used to prevent accidental ALLSTOPs made by the operator in such scripts as LOAD and LINEUP (which pause with a device out of phase).

If the **-nodeb** argument is specified, the default keyboard debouncing is /disabled/. Debounce is the default, which prevents a user from holding down ENTER from a previous command to quickly pass through the 'pse' prompt (due to typematic repeat).

For instance, to shoot a 12x wedge of a single projector frame, PSE can be used to pause before shooting each frame:

```
do 12 pse cam 1  # Wedging commands
```

The above will shoot 12x frames, prompting the operator before shooting each frame with:

```
* PAUSE *
RETURN to continue, or SPACEBAR to abort:
```

..allowing the operator to change filters, and hit return to shoot each frame. (With a filter wheel, of course, one doesn't need to do this)

To prompt for special filters each frame, make a 'wedge.run' script:

```
! echo Load ND-2 ! pse cam 1
! echo Load ND-4 ! pse cam 1
! echo Load ND-8 ! pse cam 1
[..etc..]
```

This way the camera operator will see a prompt for each filter to load.

NOTES
Whenever PSE is encountered, the keyboard's buffer is CLEARED of any previous characters to prevent accidentally typed characters from skipping several PSE commands.

SEE ALSO
RUN(OPCS) - execute a run script

ORIGIN
Gregory Ercolano, Los Feliz California 04-23-91

RAT(OPCS) Optical Printer Control System RAT(OPCS)

NAME
rat - change the projector/camera shooting ratio for REP commands

USAGE
```
rat [pro2] [pro1] [cam]    # For aerial printers
rat [pro] [cam]            # For single head printers
```

EXAMPLES
```
rat 1 1          # straight print: 1x pro1 for every 1x cam
rat 2 1          # skip print: 2x pro1 for every 1x camera
rat -2 1         # reverse skip print: -2x pro1 for 1x cam
rat 2 2 1        # aerial step print: 1x on pro1 & pro2
rat 1 -1 2       # aerial stretch frame print
```

DESCRIPTION
RAT sets the shooting ratio for any REP commands that follow.

The shooting ratio defaults to 1:1, and can be changed by the RAT command. Once set, the projector/camera ratio will remain in effect until another RAT command is executed.

You can use negative numbers in a RAT command to run in reverse.

Feet/frames specifications can be used as arguments for the RAT command. e.g.:

```
rat  1'0  3'4     # PRO shoots 1 foot, camera shoots
                  # 3 feet  4 frames
```

NOTES
Absolute specifications such as '>12' have no meaning in the context of the RAT command.

RAT does not actually SHOOT any frames, it only sets up a shooting ratio which is executed by running the REP(OPCS) command. Thus:

```
rat -2 1    # set up a -2:1 ratio
rep 12      # shoot that ratio 12 times
```

When the above example executes, the projector will shoot a total of -24 frames, and the camera will shoot 12 frames. Each time the camera shoots a frame, the projector will backspace 2 frames.

SEE ALSO
```
RAT(OPCS)           - set the shooting ratio for the REP command
REP(OPCS)           - shoot current projector/camera shooting ratio
PROPHASE(OPCSDEFS)  - sets projector phase adjustment for 1:1 shooting
MATH(DOCS)          - math expressions (for use in frame specifications)
SYNTAX(OPCS)        - Online calculator and OPCS math expression syntax
```

ORIGIN
Gregory Ercolano, Los Feliz California 11/29/89

REP(OPCS) Optical Printer Control System REP(OPCS)

NAME

rep - repeat current projector/camera shooting ratio (interlock)

USAGE

```
rep [count]                    # repeat current ratio [count] times.
    OR
rep [device] [>frame]          # repeat until [device] gets to [>frame]
```

EXAMPLES

```
rep 4          # repeat the current ratio 4 times
rep -4         # repeat the current ratio IN REVERSE 4 times
rep >4         # repeat ratio until camera gets to frame 4
rep pro >4     # repeat ratio until projector gets to frame 4
```

DESCRIPTION

The REP command shoots the current projector/camera ratio set by the last RAT command. The camera ALWAYS exposes BEFORE the projector, so when REP is executed, the frame already in the projector's gate is exposed first. Example:

rat 3 1 rep 3

...set up a 3 to 1 ratio, and shoot it 3 times. The camera FIRST shoots 1 the frame in the projector's gate, then the projector advances 3 frames... camera 1, projector 3, etc. After execution, the projector runs a total of 9 frames, the camera a total of 3 frames. This command is effectively the same as:

cam 1 pro 3	**cam 1 pro 3**	**cam 1 pro 3**
1	2	3

It's also the same as:

do 3 cam 1 pro 3

NOTES

Shooting at a ratio of 1:1 (or -1:-1) is faster than shooting with the equivalent commands cam 1 pro 1 repeatedly, because REP will run the camera and projector together.

If you use the absolute specifier '>', REP will repeat the current ratio in the direction necessary to get the camera or projectors to the specified frame. You can tell REP which motor you want to get to the specified frame by specifying any of the following after the REP command: CAM, PRO, PRO1, PRO2. Thus:

```
rep pro2 >34      # run until pro2 reaches x34
```

If you do not specify a device, the camera is always assumed:

```
rep >34           # run until camera reaches x34
```

TANDEM SHOOTING
With old mechanical printers, you had to put the projector out half phase before doing a 1:1 shoot (aka. "INTERLOCK") and then back again after you completed the shoot. You DO NOT have to worry about this when using the REP command; this is done automatically.

At the start of a 1:1 run (camera AND projector running together), the projector will stand still (remain seated) for the camera's first 1/2 rotation to get the camera and projector's movements in phase. Both motors then run together for the duration of the shoot with the movements in phase until the camera shoots its last frame. At this point the camera stops with the shutter closed (unseated), and the projector will continue moving an extra 1/2 rotation, leaving a SEATED, UNEXPOSED image in the gate, as usual.

FADE/DX
If there are any pending fades or dissolves, the fader will FIRST move to its next position before the camera exposes a frame.

FEED
If FEED(OPCS) is enabled, the motors will /first/ move to position before the camera exposes a frame. (Similar to fade/dx)

AUTOFILT
If AUTOFILT(OPCS) is enabled, the camera will first expose the filter in the gate before moving the filter wheel. (Like the projector, what you see in the gate before shooting is what's about to be shot)

ALLSTOP
Hitting the ALLSTOP key (usually the (`) key) will halt the motors at the nearest frame. ALLSTOP is safe during shooting; it will not affect exposure, and always leaves the shutter closed after stopping. The allstop key can be redefined by ALLSTOP(OPCSDEFS).

BUCKLE/VIEWER
The camera WILL NOT RUN if the buckle is tripped. When exposing film, it also won't run if the viewer is open. If either condition occurs while the camera is shooting, the motors will stop with an error at the nearest frame, similar to ALLSTOP.

If 'hardware no' is in effect, the buckle/viewer state is ignored, so that the OPCS software can run on remote computers that are not physically connected to an optical printer.

TENSION MOTORS
Before the camera or projector move, the appropriate tension motors are enabled, to ensure take-ups are in correct modes for film motion.

COUNTER OVERFLOWS
For more info on how counter overflows are handled, see CAM(OPCS).

BUGS/TODO
Needs a **-cont** flag, to continue rep commands that didn't finish shooting, ie. **rep >50 dxo 10 rep -cont >60**

SEE ALSO

RAT(OPCS)	- set the shooting ratio for the REP command
REP(OPCS)	- shoot current projector/camera shooting ratio
PROPHASE(OPCSDEFS)	- sets projector phase adjustment for 1:1 shooting
MATH(DOCS)	- math expressions (for use in frame specifications)
SYNTAX(OPCS)	- Online calculator and OPCS math expression syntax

ORIGIN
Gregory Ercolano, Los Feliz California 11/29/89

RES(OPCS) Optical Printer Control System RES(OPCS)

NAME
res - reset computer's projector/camera counters to new values

USAGE
```
res [pro2] [pro1] [cam]         # aerial, main and camera
res [pro1] [cam]                # main projector and camera
res [cam]                       # just the camera
```

EXAMPLE
```
res 0 0     # zero the projector and camera counters
res 300 -   # set pro1 counter to 300, cam counter unchanged
res 0 0 0   # zero the pro2, pro1 and camera counters
res 12'2 0  # set pro to 12 feet 2 frames, and camera to 0
```

DESCRIPTION
The RES command allows the operator to change the setting of the projector and camera counters to new values. At least 2 values must be specified.

Use '-' to leave a counter at its current value.

Feet/frame specifications are allowed as arguments, but absolute specifications (such as >34) have no meaning in the context of the RES command, as absolute positions are already implied.

SEE ALSO
RESET(OPCS) - Reset specified channels to specified values

ORIGIN
Gregory Ercolano, Los Feliz California 11/29/89

```
RESET(OPCS)             Optical Printer Control System          RESET(OPCS)

    NAME
        reset - reset the counters of one or more motors

    USAGE
        reset [chans] [position[,position,...]]

    EXAMPLES
        reset d -10             # reset position for channel D to -10
        reset efgh 0            # reset channels EFGH to ZERO
        reset efg 120,130,0     # reset e to 120, f to 130, g to 0

    DESCRIPTION
        This command allows you to specify new counter values for the
        specified channels.

    SEE ALSO
        SHOW(OPCS) - show current positions for all motors
        RES(OPCS)  - reset projector/camera counters to new values

    ORIGIN
        Gregory Ercolano, Los Feliz California  09/04/90
```

RUN(OPCS) **Optical Printer Control System** **RUN(OPCS)**

NAME
run - execute a command script

USAGE
run [filename] {optional linenumber}

EXAMPLES
```
run test.run          # start executing commands from TEST.RUN
run test.run 5        # execute test.run starting at line #5
do 12 run test.run    # execute test.run 12 times
```

DESCRIPTION
Tells the OPCS software to execute commands from a 'run' file.

Whatever commands you can type interactively can appear in a RUN file. Several commands can appear on a line, and comments can be used through-out. NEWLINES, SPACES and TABS can be used as necessary to separate lines or blocks of code.

Usually RUN(OPCS) executes the entire file. You may however specify starting execution at a particular LINE NUMBER, which is an "optional argument" that should appear after the filename. If the argument isn't supplied, '1' is the default.

The file should be an ASCII text file, and can be created by:

1) The LOG(OPCS) command
2) A text editor
3) Your own software tools

DISABLING ECHOING
Following the DOS standard, any lines in a run script file can start with '@' to prevent the line from echoing to the screen while the script executes. This is useful for preventing commands echoing to the screen that do not need to be seen by the operator. For example, the 'pse' command in the example below:

```
# INSERT ND FILTER
@pse          # wait for the operator to hit a key
```

In this case, '@' in front of 'pse' prevents the entire line from echoing to the screen. When the above is executed, one sees:

```
#  INSERT ND FILTER
Hit RETURN to continue, or ALLSTOP to abort:
```

If the '@' were removed, one would see the following more confusing output when executed:

```
# INSERT ND FILTER
pse         # wait for the operator to hit a key
Hit RETURN to continue, or ALLSTOP to abort:
```

RECURSIVE RUN COMMANDS
If the LOG(OPCS) command is in effect and a RUN command is executed, only the RUN command will appear in the log file (the LOG file will not start filling with the contents of the script that was called).

You can have run scripts call other run scripts. Keep in mind that you must adjust FILES in your DOS 'CONFIG.SYS' file to be 20 or more, depending on how many levels deep you want run scripts to call one another. These are recommended commands for your CONFIG.SYS file:

```
FILES=20
BUFFERS=40
DEVICE=ANSI.SYS
```

Scripts that call themselves, or that call parenting scripts will cause 'recursion' errors. This protects the user from creating a situation that calls itself infinitely, which would inevitably bomb out when the operating system runs out of open file handles.

LIMITS
    You can nest RUN(OPCS) commands up to 20 levels deep.

BUGS
    none yet.

SEE ALSO

DO(OPCS)	- repeat a string of commands several times
RUNCMD(OPCSDEFS)	- define built in OPCS commands as run scripts
LOG(OPCS)	- log all commands entered by the user
RUN(OPCS)	- run a log file
LOGCOUNTERS(OPCSDEFS)	- enable/disable logging counters to logfiles
LOGFORMAT(OPCSDEFS)	- formats how values are printed to logfile

ORIGIN
    Gregory Ercolano, Los Feliz California 12/16/89

SEEK(OPCS) Optical Printer Control System SEEK(OPCS)

NAME

seek - seek to positions quickly on camera/projector(s)

USAGE

```
seek [pro2] [pro1] [cam]        # slew pro2, pro1 and camera
seek [pro1] [cam]               # slew pro1 and camera
seek [cam]                      # slew just the camera
```

'-' can be used in place of arguments for motors that you do not want to change.

EXAMPLES

```
seek >101             # camera seeks to x101
seek >100  >101       # pro1 to x100, camera to x101
seek >1200 >55 >0     # pro2 to x1200, pro1 to x55, cam to x0
seek >1200 - >34      # pro2 to x1200, pro1 unchanged, cam to x34
seek 1200 1200        # slew 1200x on both pro1 and cam
```

DESCRIPTION

SEEK is used to seek to certain frame positions at high speed. This command is not to be used for exposing film (see notes below).

If SEEKCAP(OPCSDEFS) is set to 'yes', the fader will automatically cap whenever the SEEK command runs the camera. After winding, the fader will return to its previous position.

SEEK ignores any FEED(OPCS) files in progress. This allows you to shuttle film around without interfering with motion control moves.

SEEK allows a simple way to quickly seek to start positions. SEEK figures out the proper ratios to get all the motors to their positions as quickly as possible. When running the camera, the 'fastwind' speed is used (see SPD(OPCSDEFS)), instead of the current 'exposure' speed.

SEEK can be used in place of:

pro2 >1000 pro1 >1200 cam >100

..which will take a long time to run because it runs the motors one at a time. The 'seek' equivalent would be:

seek >1000 >1200 >100

Note the use of **>** to specify absolute frame positions. Without it, frame numbers will be interpreted as a relative 'windoff' value, the same way the other OPCS commands work.

If you want to ignore a particular motor, just specify a dash (-) as that motor's argument, and the motor will not be moved, i.e:

```
seek >1200 - >100
           ^
           Ignore 'projector' channel
```

Normally, camera operators will want SEEKCAP(OPCSDEFS) enabled so the fader automatically caps during a SEEK that involves camera motion to prevent exposing film.

The shutter position before SEEK executes is preserved on completion.

WARNINGS

Do not use SEEK for exposing film, it's only for slewing.
It is recommended SEEKCAP(OPCSDEFS) be enabled to avoid exposing film during seeks involving the camera.

During a seek:

> The camera runs at its slewing speed, NOT the exposure speed.
> The projectors will NOT run in sync with the camera [NOTE 1]
> The projectors may not run in sync with each other [NOTE 1]
> The state of the viewer is ignored [NOTE 2]
> Film buckles will be checked
> Any pending Fades, Dissolves, and Feeds will be unchanged
> Fader will cap to prevent exposure [NOTE 3]

NOTE #1: If their slewing speeds and/or PPR(OPCSDEFS) are different
NOTE #2: Assuming buckle and viewer are not wired together
NOTE #3: If SEEKCAP(OPCSDEFS) has been configured.

SEE ALSO

SPD(OPCSDEFS) - sets the normal and slew speeds for motors
SEEKCAP(OPCSDEFS) - configure the fader to cap during SEEK commands
PPR(OPCSDEFS) - sets Pulses Per Revolution (PPR) for each channel
MATH(DOCS) - math expressions (for use in frame specifications)
SYNTAX(OPCS) - Online calculator and OPCS math expression syntax

ORIGIN

Gregory Ercolano, Los Feliz California 09/04/90

```
SHOW(OPCS)              Optical Printer Control System          SHOW(OPCS)

    NAME
        show - show positions for all 8 motors

    USAGE
        show [-all] [-d]

    DESCRIPTION
        Shows the current position counters for all motors. Shutter motors
        (ABC) will show positions in 'frames'.

        Interpolation channels (such as the fader) will show positions in
        actual step positions, NOT interpolation positions.

        The -all flag shows all information about each motor.

        The -d is a special internal debugging flag that should
        not be used in production, and is for developer's use only.

    EXAMPLES
        >show
        A=0                   I=0
        B=0                   J=0
        C=0                   K=0
        D=0                   L=0
        E=0                   M=0
        F=0                   N=0
        G=0                   O=0
        H=0                   P=0

        >show -all
              A CHANNEL                     B CHANNEL
             Name: Aerial                  Name: Main
          Counter: 0                    Counter: 0
              PPR: 2000                     PPR: 2000
              Mrp: 500                      Mrp: 500
         NormRamp: 3,120               NormRamp: 3,120
         SlewRamp: 3,120               SlewRamp: 3,120
         ActSpeed: 0.25                ActSpeed: 0.25
        NormSpeed: 0.25               NormSpeed: 0.25
        SlewSpeed: 0.18               SlewSpeed: 0.18
              Fpf: 16                       Fpf: 16
             Slop: 0                       Slop: 0

              C CHANNEL                     D CHANNEL
             Name: Camera                  Name: Fader
          Counter: 0                    Counter: 0
              PPR: 2000                     PPR: 2000
              Mrp: 500                      Mrp: 1000

        ..snipped..

    SEE ALSO
        GO(OPCS)              - move motor channels to new step positions
        BIGCOUNTERS(OPCSDEFS) - changes on-screen counter display

    ORIGIN
        Gregory Ercolano, Los Feliz California  09/04/90
```

```
SHU(OPCS)              Optical Printer Control System          SHU(OPCS)

    NAME
        shu - move the fader to an absolute position in degrees

    USAGE
        shu [degrees]

    EXAMPLE
        shu 80.20     # move fader shutter to 80.20 degrees
        shu 0         # fully close the fader shutter

    DESCRIPTION
        Allows the operator to move the fader shutter to the absolute [degrees]
        position (in floating point degrees), and must be in the range
        0.00 to 170.00 for a 170 degree camera shutter, or
        0.00 to 120.00 for a 120 degree camera shutter.
        See INTERP(OPCSDEFS) for configuring the camera's fader settings.

    NOTES
        SHU during fades or dissolves effectively CANCELS them,
        forcing the fader to the specified [degrees] position.

        Although floating point degrees can be specified to many digits,
        actual movement will be limited to the physical resolution of the
        motor hardware. Example: If the fader is at 2.00 degrees and you
        invoke 'spd 2.01', if the resulting physical distance of such a move
        is less than a single microstep on the motor, the motor will not
        move at all, and the display will still indicate 2.00.

        The fader's floating point position is limited to 2 digits after the
        decimal point. Internally the software manages the hardware's actual
        position, which may be more accurate than what the display shows.
        So the hardware will, if capable, manage e.g. 54.000594 as a valid
        physical position internally, even if the display only shows '54.00'.

    SEE ALSO
        OPCS Commands
        CAM(OPCS)            - shoot camera (fades/dissolves too)
        OPN(OPCS), CLS(OPCS) - open/close fader shutter
        SHU(OPCS)            - move fader to an absolute position in degrees
        DXI(OPCS), DXO(OPCS) - set up dissolve in/out
        FDI(OPCS), FDO(OPCS) - set up fade in/out

        OPCSDEFS Commands
        FLOG(OPCSDEFS)   - set Fader LOGarithmic curve for custom fades
        FRANGE(OPCSDEFS) - set fade/dx's degrees range (for Hicon film stocks)
        INTERP(OPCSDEFS) - set interpolation positions (fader, focus, etc)
        SLOP(OPCSDEFS)   - correct for slop in a motor (fader, focus, etc)

        General
        MATH(DOCS)       - math expressions (for use in frame specifications)
        SYNTAX(DOCS)     - online calculator and OPCS math expression syntax

    ORIGIN
        Gregory Ercolano, Los Feliz California 11/29/89
```

SPD(OPCS) Optical Printer Control System SPD(OPCS)

NAME
spd - set the camera's exposure speed

USAGE
```
spd [speed]           # set CAMERA's exposure speed
spd [+-/*][speed]     # relative adjust current exposure speed
```

EXAMPLES
```
spd .30               # set camera's speed to .30 second
spd +.1               # add 0.1 to current camera speed
```

DESCRIPTION
This command sets the camera's exposure speed.

Speeds can also be adjusted relative to the current speed by adding a prefix to the numeric speed value with one of the following:

Prefix	Description	Examples
+	Add value to current speed	**spd +.10**
-	Subtract value from current speed	**spd -.10**
*	Multiply current speed by value	**spd *2.0**
/	Divide current speed by value	**spd /2.0**

This is useful for wedging camera exposures. EXAMPLE:

spd .20 do 9 cam 1 spd +.10

..this will shoot a 9 frame exposure wedge for the camera speeds at .10 increments, from .20 thru 1.0.

The equivalent commands without the 'do 9' loop would be:

```
spd .20 cam 1
spd .30 cam 1
spd .40 cam 1
spd .50 cam 1
spd .60 cam 1
spd .70 cam 1
spd .80 cam 1
spd .90 cam 1
spd 1.0 cam 1
```

Depending on how SPD(OPCSDEFS) is configured in OPCSDEFS.OPC, the speed values can be ROTATIONAL speed, or an EXPOSURE speed. See SPD(OPCSDEFS) for details.

SEE ALSO
```
SPD(OPCSDEFS)       - setting default speeds for motors
RAMP(OPCSDEFS)      - motor ramping maximums
MRP(OPCSDEFS)       - maximum ramping pulses for the shutter
SPDINTERP(OPCSDEFS) - set auto-interpolation for exposure speeds
```

HISTORY
The '+' prefix (e.g. 'spd +.1') was added in OPCS version K1.12E.

UNLOCK(OPCS) Optical Printer Control System UNLOCK(OPCS)

NAME
unlock - (CUSTOM) unlock motors for manual adjustment

USAGE
unlock [no arguments]

DESCRIPTION
Unlocks some or all of the motors to facilitate manual adjustment.

This command is actually a script defined with a RUNCMD(OPCSDEFS) command, and is normally customized by your local site engineer.

Some sites may prefer NOT to configure this command, since the LOAD(OPCS) and LINEUP(OPCS) prevent the need for manual adjusting.

This command normally does the following operations:

> Deenergize the motors
> Pause to allow user to manipulate the freewheeling axes
> Re-energize the motors
> Home them all

INSTALLATION NOTES
The unlock command is defined in the 'runcmd' section of OPCSDEFS.OPC:

runcmd unlock unlock.run 0

..and the file 'unlock.run' would contain something like:

```
@ # Deenergize the motors. Clearing port 379 bit #7.
@ ldefs -c clrbit 0379 80 00
# *** UNSEATED FOR LOADING ***
@ pse -noabort
@ # Re-energize motors by setting the bit.
@ ldefs -c setbit 0379 80 00
@ # Home the motors we deenergized
@ home a b c d
```

NOTE: In the older OPCS K1.xx versions 'ldefs -c' is not available, so the following commands would be used instead:

--- OLD VERSION OF OPCS --

```
@ # Deenergize the motors. Clearing port 379 bit #7.
@ ! echo @ clrbit 0379 80 00 > foo.defs ! ldefs foo.defs
# *** UNSEATED FOR LOADING ***
@ pse -noabort
@ # Re-energize motors by setting the bit.
@ ! echo @ setbit 0379 80 00 > foo.defs ! ldefs foo.defs
@ # Home the motors we deenergized
@ home a b c d
```

Note use of leading '@' signs to disable echoing of the commands, to avoid cluttering the screen with unwanted text.

ANSI characters can be added to the script to embolden messages, and erase them when the user hits a key to continue.

ORIGIN
Gregory Ercolano, Los Feliz California 04/12/98

VELREP(OPCS) Optical Printer Control System VELREP(OPCS)

NAME

velrep - special purpose velocity repeat patterns for tandem shooting

USAGE

velrep [filename] [repcnt]

'filename' is the name of a '.vrp' file (format described below) that contains the commands to define the velocities necessary for tandem shooting.

'repcnt' is the number of times to loop the velocity patterns defined in the .vrp file. (e.g. the .VRP 'repeat' command)

DESCRIPTION

This command lets advanced users define very specific velocity patterns to send to the motors for precise tandem-motor shooting, such as shooting YCM B & W separation masters at full speed.

Basically, any situation where shooting with separate cam and pro commands is too slow.

The .vrp file defines which motors will be running.

Normally this command is not executed directly by camera operators; typically a custom OPCS 'runcmd' command is defined to invoke velrep to implement shooting operations. This way, the runcmd programmer can hide the filename, which the camera operator shouldn't have to deal with.

For instance, one might define a 'ycmshoot' command in the OPCSDEFS.OPC file as:

runcmd ycm ycm.run 1

..and creating a one-line 'ycm.run' file that contains:

@velrep ycm.vrp $1

Then the operator can just type 'ycm 10', and this will actually invoke 'velrep ycm.vrp 10' behind the scenes.

LIMITATIONS

During VELREP runs, ONLY SHUTTER COUNTERS UPDATE.
Any movement on non-shutter channels (D channel and up) does NOT adjust counters. (Motor drivers in ROTCOUNT mode should count in steps for channels D and up. Or, new STEPCOUNT bit added to vels.)

The 'if_finalrep_goto" has been implemented, but not tested.

EXAMPLE

velrep campro.vrp 5 # repeat the campro.vrp pattern 5 times

```
THE .VRP FILE FORMAT
--------------------
<label>:                # text label used to identify blocks of vels
0       0       0       # vels, one for each channel (a, b, c..)
-10     0       0       # '-' prefix indicates run in reverse
10+     10+     10+     # '+' postfix indicates increment/decrement the
                        #     frame counter by 1. Inc or dec depends on the
                        #     vel's direction; 10+ will inc, -10+ will dec.
0!      0       0       # '!' postfix (in 'a' chan ONLY) does 'allstop check'
                        #     (if true, jumps to <label> for 'allstop <label>')
                        #     'allstop <label>' must be defined if ! specified.
                        #     Check is done AFTER these vels are sent to motors.
goto <label>            # where to go next after last vels sent
repeat <label>          # if repeating, go to <label>
allstop <label>         # if allstop occurs, jumps to <label> to stop the motors
                        # Only one 'allstop <label>' allowed in entire file.
                        # '!' postfix indicates where to do the allstop check.
tension +1 +1 -1 ..     # Sets directions motors are primarily running,
                        # either +1 (fwd), -1(rev), or 0 (still) for each chan.
done                    # return to OPCS, shooting completed
```

- Lines whose first character starts with a '#' are ignored.
  These are comment lines, and are not parsed by velrep.

- Each line should have no more than 256 characters.

- Always checks for BUCKLE and VIEWER as part of allstop checking.

EXAMPLE VRP FILE

```
## TEST.VRP - A simple run of projector only
##
##    > An 811 pulse ramp up/dn
##    > repeatable linear run that can be looped
##    > 'allstop/buckle/viewer' can interrupt and resume cleanly
##
##    These numbers came from running a 'pro' command where:
##
##              ppr 5000              <-- 5000 pulses per rev
##              spd b .35 .35 1 0     <-- .35 speed
##              mrp 800               <-- MRP 800
##              ramp b 10 200 10 200  <-- (not used for shutter runs)
##
##
##    To capture the vels, use 'velsav foo.pos', which can then be edited
##    using VIM columnar copy/paste. Use: MOVOP FOO.POS v2p b 1 100
##    ..to convert vels to positions to check + graph moves with GR.
##

# CHANGE TENSION MOTORS TO RUN FORWARD FOR CAM AND PRO
tension 0 +1 0

begin:
  # RAMP UP: in 811 pulses
  #
  # AER  PRO    CAM
  # ---  ---    ---
    0    1      0       # p=1
    0    3      0
    0    9      0
    0    19     0
    0    35     0
    0    61     0
    0    90     0
    0    106    0
    0    116    0
    0    122    0
    0    124    0
    0    125    0       # p=811
goto    linear_811      # Jump into middle of linear run
allstop rampdown

again:
    # Start of 10000 pulse linear frame loop
    0    135    0               # p=4189 -- this would be start of rampdown
    0    135    0
    0    136    0
    0    135    0
    0    135    0
    0    135    0               # p=5000 - would be end of rampdown

    # last 811 vels before loop
    0    136    0               # p=0 -- this would be the rampup
    0    135    0
    0    135    0
    0    135    0
    0    135    0
    0    135    0               # p=811

# Enter linear section from ramp up
linear_811:
    0    135    0
    0    135    0
    0    135    0
    0    136    0
    0    135    0
    0    135    0
    0    135    0
    0    135    0
    0    135    0
    0    135    0
    0    135    0
```

```
    0    136    0
    0    135    0
    0    135    0
    0    135    0
    0    135    0
    0    135    0
    0    135    0
    0    136+   0    # <-- '+' SUFFIX: ADVANCE PRO COUNTER +1
    0    135    0
    0    135    0
    0    135    0
    0    135    0
    0    135    0
    0!   135    0    # p=4189  <-- ALLSTOP CHECK
                     #             This is where we jump to 'rampdown'
                     #             if allstop or buckle/viewer is tripped

repeat again         # Repeat loop:
                     # Falls thru to "rampdown" on last frm of VRP repeat count

rampdown:
    # Here's the 811 pulse rampdown to a stop.
    0    125    0    # p=(4189+125)
    0    124    0
    0    122    0
    0    116    0
    0    106    0
    0    90     0
    0    61     0
    0    35     0
    0    19     0
    0    9      0
    0    3      0
    0    1      0    # p=5000 (STOP)
    0    0      0    # ** STOP **
    0    0      0    # <- HOLD for several samples (debugging)
    0    0      0
    0    0      0
    0    0      0
    0    0      0
    0    0      0
    0    0      0
    0    0      0
    0    0      0
done
```

ORIGIN

Gregory Ercolano, Altadena, California 12/15/03

VELSAV(OPCS) Optical Printer Control System VELSAV(OPCS)

NAME

velsav - special purpose save motor run velocities to file

USAGE

velsav <filename> -- starts saving motor velocities to <filename>
velsav off -- turn off saving vels

EXAMPLE USE

velsav vels.txt -- enables motor vels to be saved to vels.txt
go abc 100,200,500 -- moves motors (adds content to vels.txt)
velsav off -- closes vels.txt file
edit vels.txt -- view contents of vels.txt to see vels

DESCRIPTION

This command allows motor velocities used during motor moves to be saved (appended) to the specified filename.

VELSAV(OPCS) is similar to LOG(OPCS), staying in effect until disabled with '**velsav off**'.

If enabled, whenever a motor movement command (like 'go' or 'cam') completes, the motor velocities still in the ring buffer from the move are appended to the specified file. Example for a 'go' command:

```
# go abc -50,-100,-200
#----------- VELS ------------ # ---------------- DISTANCE ---------------
#  A    B    C    D    E    F  #    A     B      C      D      E      F
#---- ---- ---- ---- ---- ---- # ------ ------ ------ ------ ------ ------
 -10  -10  -10  0    0    0    # -10    -10    -10    0      0      0
 -20  -20  -20  0    0    0    # -30    -30    -30    0      0      0
 -10  -30  -30  0    0    0    # -40    -60    -60    0      0      0
 -10  -20  -40  0    0    0    # -50    -80    -100   0      0      0
 0    -10  -30  0    0    0    # -50    -90    -130   0      0      0
 0    -10  -30  0    0    0    # -50    -100   -160   0      0      0
 0    0    -20  0    0    0    # -50    -100   -180   0      0      0
 0    0    -10  0    0    0    # -50    -100   -190   0      0      0
 0    0    -10  0    0    0    # -50    -100   -200   0      0      0
 0    0    0    0    0    0    # -50    -100   -200   0      0      0
```

Each line has the velocity samples for the channels of the move, followed on the right by the total distance traveled for each channel.

The LEFT 6 COLUMNS are the raw motor velocities (in decimal values) for each of the 6 motor channels A-F.

The RIGHT 6 COLUMNS shows the accumulated distance traveled (in decimal value steps) since the beginning of the move.

> KUPER/RTMC cards use 120 vels per sec, so each row is 1/120th sec.
> A800 cards use 107 vels per sec, so each row is 1/107th sec.

The data saved by this command can be used to build VELREP(OPCS) files by copy/pasting together the columns of velocities.

LIMITATIONS

Not all motor movement commands support having velocities saved, e.g. VELREP(OPCS), FEED(OPCS).

The output is designed to fit on a DOS 80 column screen for easy editing. So only channels A-F are saved.

Moves longer than 15 seconds will show up truncated. Only what's left behind in the motor's 64k ring buffer is saved, which at 32 bytes per sample and 120 samples per second means a limit of 2048 samples, or about 16 seconds of motor movement.

This command will reveal the inner complexities of some commands, like REP and SEEK, which do small movements before and after the main movement. For instance, REP will phase-adjust the projector before and after a tandem run. These small movements will be included in the file as separate tables of movement.

HISTORY

Added in K2.10TC 02/14/2022.

ORIGIN

Gregory Ercolano, Alhambra, California 02/14/22

This page intentionally left blank.

DEFS COMMANDS

```
INTRO(OPCSDEFS)         Optical Printer Control System          INTRO(OPCSDEFS)

    The OPCSDEFS section of the manual describes the OPCSDEFS.OPC config
    file commands, generally used only during initial setup of the software.
    To get a list of all the OPCSDEFS commands, use: 'man -k OPCSDEFS:', e.g.

       !            - DOS shell execute
       allstop      - define the ALLSTOP key
       bang         - DOS shell execute
       baseaddr     - sets motion control card's base address
       bigcounters  - enable/disable big counters
       buckle       - define buckle sensor ports/bits
       clrbit       - clear bit(s) on a port
       debugger     - enable OPCS system debugging
       dirxor       - invert a motor's direction
       doscmd       - define a DOS command to OPCS
       faderdisplay - on/off display of fader counter
       filter       - define channel to control a filter wheel
       flog         - set fader logarithm characteristics
       fpf          - set the 'frames per foot' for a motor
       frange       - set fade/dx's degree range (Hicon film stocks)
       hardware     - enable/disable hardware
       interp       - set up interpolations for a motor
       jogstep      - set jog mode's vernier/crawl stepping rate
       keyfunc      - define keys for KEY(OPCS) mode
       logcounters  - update counter info to log files
       logformat    - format how counters appear in log files
       mrp          - set maximum ramp pulses for shutter ramping
       opcscmd      - execute an OPCS command from within defs file
       ppr          - set the pulses per revolution
       pro2display  - on/off display of pro2 info
       prophase     - set pro phase adjust for 1:1 shooting
       ramp         - set motor's max accels & velocities
       rampcurve    - sets ramping curve for shutter runs
       respond      - setup device for responses to commands
       runcmd       - define custom OPCS commands
       sampspersec  - sets the velocity sample rate
       setbit       - set bit(s) on a port
       seekcap      - set auto-fader cap for seek command
       slop         - set the amount slop in fader
       spd          - set the initial speed for a channel
       spdinterp    - set auto-interpolation for exposure speeds
       startspeed   - start speed for ramping
       tension      - set the tension motor port
       tripswitch   - set up tripswitches
       viewer       - set the viewer input port and bit mask
       xorbit       - flip bit(s) on a port (exclusive-or)

SEE ALSO
        OPCS(DOCS) - OPCS main program
      SYNTAX(DOCS) - Online calculator and OPCS math expression syntax
        A800(DOCS) - Notes on the A800 stepper motor control board
      RTMC48(DOCS) - Notes on the RTMC48 stepper motor control board
     CENTENT(DOCS) - Centent driver wiring
    QUICKREF(DOCS) - Camera operator quick reference       ***
    OPCSETUP(DOCS) - OPCS hardware/software setup details  ***
    OPCSHARD(DOCS) - OPCS hardware                         ***
     VERSION(DOCS) - OPCS version history

ORIGIN
    Gregory Ercolano, Los Feliz California 08/17/20
```

```
ALLSTOP(OPCSDEFS)       Optical Printer Control System    ALLSTOP(OPCSDEFS)

    NAME
        allstop - define the ALLSTOP key

    USAGE
        allstop [port] [mask] [test] [0]    # (all values in hex!)

    EXAMPLES
        allstop 0060 7f 46 0                # typical for SCROLL LOCK key

    DESCRIPTION
        This command defines the allstop key for the software. The ALLSTOP
        button can be any key, or for that matter any bit on any port on the
        IBM PC, but keyboard's (`) key is recommended.

        All parameter values are in hexadecimal.

            [port] is the port number to read in the range 0000-03ff.
            Use 0060 for the keyboard.

            [mask] is applied to the value received from the port
            whenever the software is checking for an allstop condition.
            This is applied before comparing to [test].

            [test] is compared to the value read from the port after
            [mask] is applied. If the result is the same as [test], an allstop
            condition exists.

            [0] is always zero.

        Under normal conditions, [port] is 0060 (the keyboard port), [mask]
        is usually '7f' (meaning mask off the high bit) and [test] is normally
        '29', which is the (`) key's keyboard scancode.

        The software essentially uses the following C code to read the port:

            if ( ( inp(port) & mask ) == test )
            {
                /* An allstop condition exists  */
            }

    BUGS
        None.

    SEE ALSO
        DEENERGIZE(OPCSDEFS)  - define port/bit to deenergize motors
        ALLSTOP(OPDSDEFS)     - define port/bit to detect the allstop key
        BUCKLE(OPCSDEFS)      - define port/bit to detect film buckles
        VIEWER(OPCSDEFS)      - define port/bit to detect viewer open
        TRIPSWITCH(OPCSDEFS)  - define port/bit to detect trip switches
        SETBIT(OPCSDEFS)      - set bit(s) on a port
        CLRBIT(OPCSDEFS)      - clear bit(s) on a port
        XORBIT(OPCSDEFS)      - invert bit(s) on a port

    ORIGIN
        Gregory Ercolano, Los Feliz California 09/11/90
```

!(OPCSDEFS) Optical Printer Control System !(OPCSDEFS)

NAME

! - execute a shell command from within OPCSDEFS file

SYNOPSIS

! [command]

EXAMPLES

```
! copy *.* a:\safe        # run the DOS 'copy' command
! myprog 12 34 56         # run 'myprog 12 34 56' from DOS
```

DESCRIPTION

Like the !(OPCS) command, !(OPCSDEFS) executes commands from DOS in a DOS shell. But this allows the user to have programs executed automatically while the OPCSDEFS.OPC file is being parsed when the OPCS software is started.

This is useful to initialize other pieces of hardware not associated with the OPCS software, or other modular programs that should be run as part of the initialization process.

COMMAND STACKING

As with all OPCSDEFS commands, it is recommended that you avoid 'stacking' more than one command on any one line (like you can with OPCS commands at the operator's prompt). Put each separate OPCSDEFS command on its own separate line to avoid parsing problems.

BUGS

'!' commands must be on a line by themselves. So you can't 'quote' commands with '!' in DEFS files the way you can at the OPCS prompt. So for example, the following will NOT WORK in an OPCSDEFS file:

```
! myprog 12 34 56 ! ramp a 1 12
```

ORIGIN

Adapted after examples set by such UNIX utilities as VI, ED, etc. UNIX is a trademark of AT&T.

BASEADDR(OPCSDEFS) Optical Printer Control System BASEADDR(OPCSDEFS)

NAME

baseaddr - configure the Kuper/A800 motion control card base address

NOTE: This command was removed in OPCS K2.10 and up.
See OBSOLETE and HISTORY below.

USAGE

```
baseaddr [port]    # port value in hex, so "300" is 300 hex
```

EXAMPLES

```
baseaddr 300       # sets base address to 300 hex (default)
```

DESCRIPTION

The stepper motor pulse generator motion control cards that OPCS uses all have a "base address" that is configured with jumpers on the cards themselves.

The 'baseaddr' must match the jumper setting on the card.

The default for all the cards is '300', a commonly available hex address for industrial cards.

OBSOLETE

In OPCS K2.10 (and up) the 'baseaddr' command was made obsolete, and should no longer specified in OPCSDEFS.OPC, in favor of setting the card's base address as a command line parameter to the driver, e.g.

```
a800drv.com -b0300 -- starts A800 driver, sets baseaddr to 300
rtmc48.com -b0300  -- starts RTMC48 driver, sets baseaddr to 300
mdrive.com -b0300  -- starts RTMC16 driver, sets baseaddr to 300
```

See "HISTORY" below for the history of this change.

BASE ADDRESS CONFIGURATION

For information on how to configure the stepper pulse generator cards for different "base address" and IRQ settings, see the following documents:

```
man a800   -- the A800 card documentation
man rtmc16 -- the RTMC16 card documentation
man rtmc48 -- the RTMC48 and Kuper Industrial card documentation
```

HISTORY

This command was introduced in K2.00, and removed in K2.10:
In K1.xx the base address was configured in the "STARTUP.DEFS" file.
In K2.00 "STARTUP.DEFS" was removed, and the command moved to OPCSDEFS.OPC.
In K2.10 "baseaddr" was removed from OPCSDEFS.OPC in favor of setting the base address as a command line parameter to the card's driver.
(See "OBSOLETE" above for info on how that's specified)

SEE ALSO

```
A800(DOC)   - Documentation for the A800 card and driver
RTMC48(DOC) - Documentation for the RTMC48 card and driver
RTMC16(DOC) - Documentation for the RTMC16 card and driver
```

ORIGIN

Version K2.00 Gregory Ercolano, Alhambra California 05/09/20

BIGCOUNTERS(OPCSDEFS) Optical Printer Control System BIGCOUNTERS(OPCS)

NAME

bigcounters - sets the style of the onscreen counters

SYNOPSIS

```
bigcounters on       # big counters (same as 'large')
bigcounters off      # small font counters (same as 'mocon')
bigcounters large    # (K2.xx) big counters, takes 17 lines on display
bigcounters nixie    # (K2.xx) "nixie" counters, maximizes digits
bigcounters mocon    # (K2.xx) small font counters, showing ALL mocon channels
bigcounters small    # (K2.xx) small font counters, takes 3 lines on display
```

DESCRIPTION

This command controls the style of counter display you see at the top of the OPCS screen.

In OPCS K1.xx the only options were 'on' and 'off'.
The other options were added in K2.xx and up.

The best display for simple printing is either '**bigcounters on**' (default) or '**bigcounters nixie**', the latter a slightly smaller size allowing for more digits per counter, but both sizes are easy to read from across a room.

For motion control moves, it's best to have '**bigcounters mocon**', which shows the small printer counters, and the positions of all channels A-P. This way while axes are moving, you can monitor all their positions in real time.

The absolute smallest counters are '**bigcounters small**', which take up only the top 3 lines of the screen, leaving the rest for the camera operator's command history.

With one of the above commands in your OPCSDEFS.OPC file, the system will always start up with the mode you prefer.

COUNTER OVERFLOWS

For more info on how counter overflows are handled, see CAM(OPCS).

HISTORY

In K1.xx, only 'on' and 'off' were available, which are the equivalent of 'large' and 'mocon' respectively in K2.00 and up.

In version K2.00 the options "small", "nixie", "mocon" and "large" were added to make it easier to specify the 4 different counter styles.

BUGS/LIMITATIONS

See above regarding display counter overruns.

In OPCS K1.xx and older, errors detected in OPCSDEFS files do not print the line number on which the error was detected.
In K2.00 and up, errors include the line number.

ORIGIN

Gregory Ercolano, Los Feliz California 12/15/89

BUCKLE(OPCSDEFS) Optical Printer Control System BUCKLE(OPCSDEFS)

NAME

buckle - configure the buckle input ports and bit masks

USAGE

buckle [port] [mask] [test] [0] # (all values in hex!)

EXAMPLES

```
buckle c 0000 00 00      # No buckle detection for camera chan
buckle c 03bd 40 40      # LPT1 pin 10, HI bit detects condition
buckle c 03bd 40 00      # LPT1 pin 10, LO bit detects condition
```

DESCRIPTION

If your system has a buckle sensor switches on the camera or projectors, they can be wired to one of the IBM parallel ports to allow the software to sense its state.

A buckle sensor temporarily stops shooting to prevent film jams, should the film deviate from its normal path and "buckle" into the sensors.

Buckle conditions are checked whenever a shooting command is executed such as KEY, CAM, REP and SEEK. Buckle conditions are NOT tested when linear movement commands such as with GO(OPCS), JOG(OPCS).

All parameter values are in hexadecimal.

[port] is the port number in the range 0000-03ff.
If [port] is 0000, no buckle checking is done for that channel.

[mask] is applied to the value received from the port whenever the software is checking for a viewer open condition. This is applied before comparing to [test].

[test] is compared to the value read from the port after [mask] is applied. If the result is the same as [test], a viewer open condition exists.

WIRING CONSIDERATIONS

Normally you would use an optically isolated interface card for the buckle and viewer switches, e.g. PIO-100(DOCS).

If directly connecting switches to the parallel port, use a separate dedicated 5 VDC power supply wired through the switches such that when the sensing switch is tripped, +5V is supplied to the computer.

Such a supply can be a store-bought 5 VDC regulated switching transformer, which gives out *exactly* 5VDC. (Avoid older linear transformers, as they are often unregulated)

As with any signal going to the sensing input on a computer, the signal should never be open. The signal must pull either 5 volts or ground for a TRUE or FALSE condition. Open inputs can act like radio antennas that will oscillate randomly.

If you have noise problems, shielded wire can help mitigate noise. Shield ONLY at the power supply end. Ground the shield to chassis ground if possible. Keep wire lengths as short as possible.

If noise problems persist, it may be that the wire is simply too long for such a low voltage signal. The PIO-100 uses 12VDC to allow for longer cable lengths, and an optoisolator to convert to +5V for the parallel port input.

If noise problems persist, and you have ruled out the computer, it may be that the wire is simply too long for such a low voltage signal. You may want to use a higher voltage (12 volts) in the switch circuitry to drive an optoisolator close to the computer, using the optoisolator to switch a 5 volt current to the port.

You can find the base port value for the parallel ports from the operating system using the DOS 'debug' utility:

```
C>debug                                 # run 'debug'
-d40:8 f                                # enter this (not the '-')
0040:0008  BC 03 78 03 00 00 00 00      # debug spits this out
-q         ----- -----                  # type 'q' to quit debug
             |     |
             |     LPT #2's  port base address
             |
             LPT #1's  port base address
```

Your machine may show different values. In the case above, 03BC is the base port value for LPT1..note the bytes are in reverse order in typical LSB/MSB fashion.

See the PARALLEL() man page which shows the pin out and port addresses of the IBM PC's parallel ports.

BUGS

None.

SEE ALSO

DEENERGIZE(OPCSDEFS)	- define port/bit to deenergize motors
ALLSTOP(OPDSDEFS)	- define port/bit to detect the allstop key
BUCKLE(OPCSDEFS)	- define port/bit to detect film buckles
VIEWER(OPCSDEFS)	- define port/bit to detect viewer open
TRIPSWITCH(OPCSDEFS)	- define port/bit to detect trip switches
SETBIT(OPCSDEFS)	- set bit(s) on a port
CLRBIT(OPCSDEFS)	- clear bit(s) on a port
XORBIT(OPCSDEFS)	- invert bit(s) on a port
PARALLEL(DOCS)	- parallel port pinout with port/bit masks
PIO-100(DOCS)	- OPCS Parallel I/O interface board, e.g. http://seriss.com/opcs/pio-100/

ORIGIN

Version K1.12e+ Gregory Ercolano, Venice California 04/11/98

```
CLRBIT(OPCSDEFS)        Optical Printer Control System    CLRBIT(OPCSDEFS)

    NAME
        clrbit - clear bit(s) on an IBMPC port

    USAGE
        clrbit [port] [mask] [softlatch]    # (values hex!)

    EXAMPLES
        clrbit 0378 04 0        # lpt1 port 0378, bit #2 (0x04)
        clrbit 0306 01 1        # kuper logic connector softlatch bit #1

    DESCRIPTION
        This command disables bits on a port based on a bit mask.
        All bits specified in the mask are cleared. All values are in hex.

            [port] is the port number in the range 0000-03ff

            [mask] is a hex byte value indicating the bits
            to be cleared on that port.

        SETBIT(OPCSDEFS) and CLRBIT(OPCSDEFS) can be used in OPCSDEFS.OPC
        to initialize port hardware bits to known states on OPCS startup.

    CAVEATS
        o With [softlatch] set to 1, only ports 0x0000 - 0x07ff are allowed.
          Any ports above 0x07ff with [softlatch] enabled causes an error.

        o External programs changing port bits defined to OPCS with [softlatch]
          (e.g. the kuper logic I/O port) should be aware that OPCS is
          maintaining its own internal latch for that port, and that latch
          won't know about hardware changes made by external programs.

        o Due to these issues, it's best to avoid using hardware that has to
          be latched. It's usually bad hardware practice to make WRITE ONLY
          ports, since different programs cannot co-communicate with them,
          unless some common data area or driver is arranged.

    SEE ALSO
        DEENERGIZE(OPCSDEFS)  - define port/bit to deenergize motors
        ALLSTOP(OPDSDEFS)     - define port/bit to detect the allstop key
        BUCKLE(OPCSDEFS)      - define port/bit to detect film buckles
        VIEWER(OPCSDEFS)      - define port/bit to detect viewer open
        TRIPSWITCH(OPCSDEFS)  - define port/bit to detect trip switches
        SETBIT(OPCSDEFS)      - set bit(s) on a port
        CLRBIT(OPCSDEFS)      - clear bit(s) on a port
        XORBIT(OPCSDEFS)      - invert bit(s) on a port

    ORIGIN
        Version K1.12d+ Gregory Ercolano, Venice California 03/04/98
```

```
CMDLINE(OPCSDEFS)      Optical Printer Control System         CMDLINE(OPCSDEFS)

    NAME
        cmdline - set the OPCS command line editing style

    SYNOPSIS
        cmdline [dos|editor]

            dos    - Old MS-DOS style command line editing (F1, F3, F5..)
            editor - New interactive command line editing with history

    EXAMPLES
        cmdline dos     -- old style command line editing (pre-K200)
        cmdline editor  -- enable new interactive command line editor

    DESCRIPTION
        New in K200 and up, this command lets you change the OPCS command
        line to respond to interactive text editing keystrokes, allowing
        Insert/Delete and Left/Right arrows for line editing,
        and Up/Down arrow keys for command history.

        'cmdline dos'
        -------------
        This is the very old MS DOS style line editing using F2/F3/F5 function
        keys, and left/right arrow keys. It's basically whatever the default
        MS-DOS command line editing provides.

        'cmdline editor'
        ----------------
        This is the new editor style, which supports more interactive line
        editing that people are generally familiar with in interactive
        text editors like MS-DOS 'EDIT', NOTEPAD, etc. Edit keys supported:

             Up Arrow -- previous line in command history      (^P)
             Dn Arrow -- next line in command history          (^N)
             Lt Arrow -- move reverse one char on current line (^B)
             Rt Arrow -- move forward one char on current line (^F)
            Backspace -- backspace and delete                  (^H)
               Delete -- delete character                      (^D)
                 Home -- move to start of current line         (^A)
                  End -- move to end of current line           (^E)
            Ctrl-Home -- jump to top of command history
             Ctrl-End -- jump to bottom of command history (current line)
            Ctrl-Left -- word left
           Ctrl-Right -- word right
                   ^K -- clear to end of line
                   ^U -- clear current line (hit again to 'undo')
                   ^V -- enter next character literally
                  ESC -- clear current line (hit again to 'undo')
                   F3 -- re-type last command
                   F4 -- re-run last command (F3 + Enter)

    SEE ALSO
        OPCS(DOCS) -- See section on COMMAND LINE EDIT KEYS
        QUICKREF(DOCS) -- Camera operator tutorial covers editing keys

    ORIGIN
        Gregory Ercolano, Los Angeles, California 08/21/2020
```

```
DEBUGGER(OPCSDEFS)      Optical Printer Control System       DEBUGGER(OPCSDEFS)

    NAME
        debugger - enable OPCS debugging

    SYNOPSIS
        debugger [value]

    EXAMPLE
        debugger 3    # enable somewhat heavy debugging
        debugger 0    # disable all debugging

    DESCRIPTION
        Intended only for debugging the OPCS software during development.
        To view velocities for motor runs, use VELSAV(OPCS) instead.

    EXAMPLE USE
        To capture the voluminous data, start OPCS with output redirected
        to a file, blindly run the commands to test, then quit. Example:

            C:\> opcs > output.log
            ldefs -c debugger 3         <-- type this and hit return
            pro2 5                      <-- type this and hit return
            qq                          <-- type this and hit return
            C:\> more output.log        <-- view the debugger output

        Look for "POST MORTEM" in the output which will show a hexdump
        of motor runs. Example:

              A    B    C    D    E    F    G    H     <-- Channels
              |    |    |    |    |    |    |    |
POST MORTEM \|/  \|/  \|/  \|/  \|/  \|/  \|/  \|/
56eb-0000: 000b 0000 0000 0000 0000 0000 0000 0000  \
56eb-0020: 0059 0000 0000 0000 0000 0000 0000 0000   |
56eb-0040: 00a6 0000 0000 0000 0000 0000 0000 0000   |
56eb-0060: 00b2 0000 0000 0000 0000 0000 0000 0000   | Hex dump
56eb-0080: 00df 0000 0000 0000 0000 0000 0000 0000   | of Ring Buffer
56eb-00a0: 00de 0000 0000 0000 0000 0000 0000 0000   |
56eb-00c0: 00de 0000 0000 0000 0000 0000 0000 0000   :       __ Indicates
56eb-00e0: 00de 0000 0000 0000 0000 0000 0000 0000          /   ROTATION bit
56eb-0100: 50df 0000 0000 0000 0000 0000 0000 0000  <-AS <-ROT  Set (in 'A')
56eb-0120: 00de 0000 0000 0000 0000 0000 0000 0000     \
56eb-0140: 00de 0000 0000 0000 0000 0000 0000 0000      \__ Indicates ALLSTOP
56eb-0160: 00de 0000 0000 0000 0000 0000 0000 0000          Bit Set (in 'A')
56eb-0180: 00de 0000 0000 0000 0000 0000 0000 0000
56eb-01a0: 00df 0000 0000 0000 0000 0000 0000 0000
56eb-01c0: 00de 0000 0000 0000 0000 0000 0000 0000
56eb-01e0: 00de 0000 0000 0000 0000 0000 0000 0000
56eb-0200: 00de 0000 0000 0000 0000 0000 0000 0000
56eb-0220: 50df 0000 0000 0000 0000 0000 0000 0000  <-AS <-ROT
56eb-0240: 00b2 0000 0000 0000 0000 0000 0000 0000  <-ASADDR
56eb-0260: 00a6 0000 0000 0000 0000 0000 0000 0000      \__ Indicates ALLSTOP
56eb-0280: 0059 0000 0000 0000 0000 0000 0000 0000          Address
56eb-02a0: 000b 0000 0000 0000 0000 0000 0000 0000
56eb-02c0: 0000 0000 0000 0000 0000 0000 0000 0000
56eb-02e0: 8000 8000 8000 8000 8000 8000 8000 8000
 TOTAL     0fa0 0000 0000 0000 0000 0000 0000 0000  <- total distance in hex
 TOTAL     4000 0    0    0    0    0    0    0     <- total distance in dec

    WARNING
        When debugging is enabled, software may not control hardware
        correctly due to the volume of data generated.
```

DIRXOR(OPCSDEFS) Optical Printer Control System DIRXOR(OPCSDEFS)

NAME

dirxor - invert direction of motor; direction "exclusive or" (XOR)

SYNOPSIS

dirxor [chan] [invert]

EXAMPLE

```
dirxor b 1    # invert direction of the projector motor.
dirxor c 1    # invert direction of the camera motor.
```

DESCRIPTION

After setting up a new motor, you notice that executing a command that should run the motor forward actually runs it BACKWARDS, and vice-versa. You can remedy this by either rewiring the motor (the hard way), or changing the DIRXOR(OPCSDEFS) command in your OPCSDEFS.OPC file.

By changing the **[invert]** value from 0 to 1 (or 1 to 0, as the case may be) you can invert the software's sense of direction for that motor.

[chan] is the motor channel being affected.

[invert] can be either a 0 (default) or a 1, depending on which value suits the installation.

BUGS

None.

ORIGIN

Gregory Ercolano, Los Feliz California 11/29/89

DOSCMD(OPCSDEFS) Optical Printer Control System DOSCMD(OPCSDEFS)

NAME
doscmd - define DOS commands that don't need the '!' prefix

USAGE
doscmd [command]

EXAMPLES
doscmd dir # 'dir' command doesn't need the '!' prefix
doscmd man # 'man' command
doscmd -clear # clears all 'doscmd' definitions (K2.02+)

DESCRIPTION
This command allows execution of commonly used DOS commands within OPCS without the need for using the '!' prefix.

Assuming 'doscmd dir' has been defined, you can then type 'dir' or 'dir *.run' or 'dir *.run /w' without needing the '!' prefix.

When the command you defined is invoked from OPCS, all arguments to the right of the command up to the end of line are passed as arguments to the DOS command. Comment characters ('#') can be used:

dir *.run /w | more # comment text can appear here

In the above example, 'dir *.run /w | more' is passed to DOS, and the comment character and text will be ignored.

The **-clear** option clears all previous doscmd definitions.

LIMITS
The user may define up to 30 such commands, each command having a 10 character limit. The command itself should be in the current directory or in the execution PATH if it is an 'EXE' or 'COM' program. Refer to your DOS manual for setting execution paths.

GOTCHYAS
This command can seem really nice at first, but it does have drawbacks.

- o RUN scripts containing commands that are really DOSCMD definitions will fail on other OPCS systems that don't have the same DOSCMD definitions. For portability, use the '!' prefix instead of assuming DOSCMD's have been defined.
- o ALL arguments to the right of the DOS command will be passed as arguments for the command. This means you cannot 'stack' other DOS or OPCS commands on the same line. If you really need to stack commands after a DOS command, use "!" instead.

HISTORY
The -clear option was added in K2.02

SEE ALSO
!(OPCS) - execute a dos command
RUNCMD(OPCSDEFS) - define your own OPCS command as a RUN script

ORIGIN
Gregory Ercolano, Los Feliz California 04/08/91

```
ECHO(OPCSDEFS)      Optical Printer Control System       ECHO(OPCSDEFS)

    NAME
        echo - enable/disable echoing of certain defs commands

    USAGE
        echo [on|off]

    EXAMPLES
        echo on       # enable echoing
        echo off      # disable echoing

    DESCRIPTION
        Some opcsdefs commands (such as '!') echo messages to the screen.
        In cases where this is not desirable, 'echo off' can be used
        to disable this.

        Also, '@' can be used to prefix any command, to disable its echoing.
        '@' only affects the command it precedes.

    ORIGIN
        Added K1.12d+ Gregory Ercolano, Venice California 04/09/98
```

FADERDISPLAY(OPCSDEFS) Optical Printer Control System FADERDISPLAY(OPCSDEFS)

NAME
faderdisplay - Enable/Disable display of fader position counter

USAGE
faderdisplay [on|off]

EXAMPLES
faderdisplay on # show fader position
faderdisplay off # don't show fader position

DESCRIPTION
Some printer systems do not have motorized faders. This command disables the fader counter to avoid confusing camera operators.

ORIGIN
K1.12f+ Gregory Ercolano, Venice California 05/14/98

FILTER(OPCSDEFS) Optical Printer Control System FILTER(OPCSDEFS)

NAME

filter - define the channel that controls the filter wheel

USAGE

filter [chan] [tvels] [vels]

EXAMPLES

filter h 8 4 10 16 20 20 16 10 4

```
       | |  ---------------------
       | |               |
       | |               The 8 velocity samples sent to the motor,
       | |               including ramping. Sum is 100.
       | |
       | 8 velocity samples
       |
       Channel with filter wheel
```

DESCRIPTION

This command defines which channel controls a 'wedge wheel', or 'filter wheel'. These settings configure the AUTOFILT(OPCS) command.

Most filter wheels usually have 20 filters. If the controlling motor is 2000 pulses per revolution, and the motor is geared 1:1, then it would take 100 pulses to move the motor 1 filter position.

[chan] is the channel controlling the filter wheel.

[tvels] is the number of velocity samples specified.

[vels] are the velocity samples. These values are the raw speed values (velocities) sent to the motor. These values should smoothly ramp up then back down. The sum of these values should be the number of pulses to move to the next filter position.

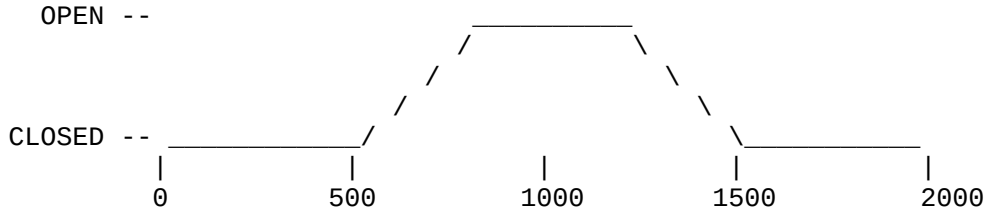
These values are configured by hand, since the idea is to get the motor to move to the next position in as little time (as few values) as possible, without stalling the motor. At the fastest camera speed, you will want the filter wheel to be at rest while the camera is exposing film.

NOTES

When AUTOFILT(OPCS) is enabled, the filter wheel will begin moving after each camera frame exposes, at the moment the camera shutter has completely closed. This is determined by the MRP(OPCSDEFS) value.

If you find the filter wheel starts its movement while the shutter is still open, you should either decrease the MRP(OPCSDEFS) value, or make sure your home position for the shutter is accurate.

The following diagram shows the travel of the camera shutter through a full rotation, from left to right. The numbers indicate the pulses throughout the rotation of a shutter that is 2000 pulse per rev:



If MRP(OPCSDEFS) is 500, then the AUTOFILT command will begin moving the filter wheel at the 1500 position, ie. 2000 - 500. The shutter should be fully closed at that position.

If you decrease the MRP(OPCSDEFS) value to 400, AUTOFILT will begin moving the filter wheel at the 1600 position, i.e. 2000 - 400.

SEE ALSO

AUTOFILT(OPCS) - enable auto-wedging with a filter wheel
MRP(OPCSDEFS) - maximum ramp pulses for shutter motors

ORIGIN

Version K1.12e+ Gregory Ercolano, Venice California 04/10/98

FLOG(OPCSDEFS) Optical Printer Control System FLOG(OPCSDEFS)

NAME

flog - configure fader's logarithmic movement

SYNOPSIS

```
flog [value]    # value is a signed floating point number
                # (-2.0 is recommended for NORMAL fades)
```

DESCRIPTION

Sets characteristics of log function used to calculate fades. This command lets you harden the log curve, or 'soften' it to almost linear.

Values for FLOG are normally negative, since it is usually desirable to have a sharper curve on the OPEN end of a fade.

In the NEGATIVE range, FLOGs that approach -1.0 force the curve to become more linear, whereas values that approach -10000 (lowest value recommended) harden the fading curve on the OPEN end (normal).

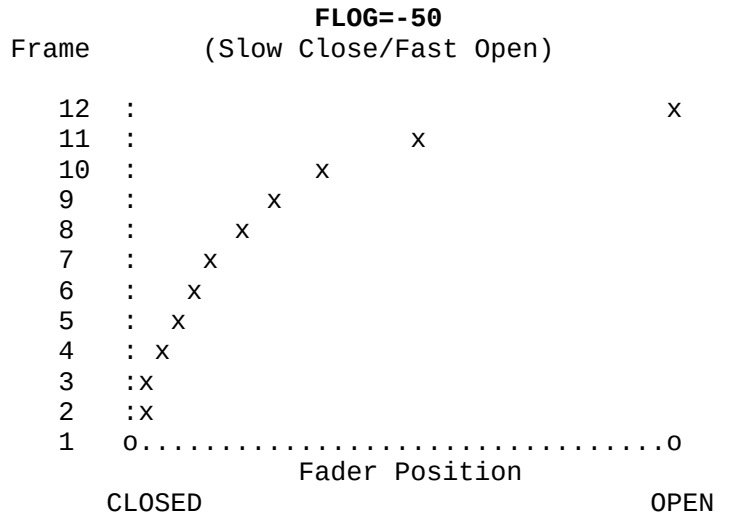
In the POSITIVE range, FLOGS that approach 1.0 force the curve to become more linear, whereas values that approach 10000 (highest value recommended) harden the fading curve on the CLOSED end (giving a comical 'zip' to/from black).

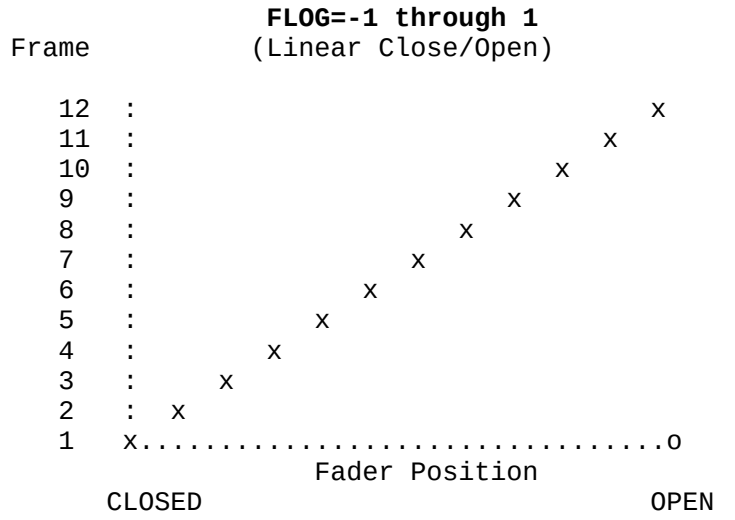
Values between -1.0 and 0.0 (and values between 1.0 and 0.0) cause a linear fade, which is more or less a DISSOLVE.

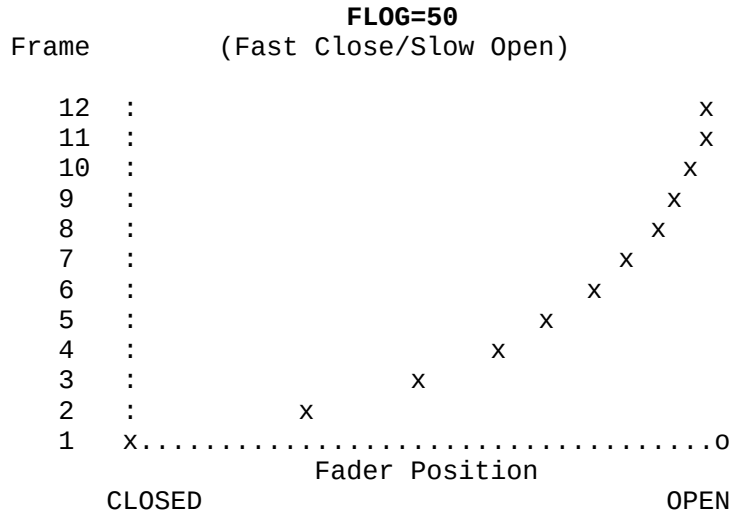
TABLE OF FLOG VALUES AND THEIR EFFECT ON A 12 FRAME FADE

	<---- FLOG VALUES ---->							
FRM	-50	-20	-5	-2	2	5	20	50
1	3.70	4.68	7.29	10.44	19.63	30.39	53.86	70.66
2	7.75	9.78	15.12	21.34	37.81	53.96	80.99	96.28
3	12.21	15.39	23.57	32.75	54.73	73.22	99.26	112.29
4	17.19	21.61	32.76	44.72	70.56	89.50	113.07	123.96
5	22.81	28.60	42.83	57.30	85.43	103.60	124.16	133.16
6	29.26	36.57	53.96	70.56	99.44	116.04	133.43	140.74
7	36.84	45.84	66.40	84.57	112.70	127.17	141.40	147.19
8	46.04	56.93	80.50	99.44	125.28	137.24	148.39	152.81
9	57.71	70.74	96.78	115.27	137.25	146.43	154.61	157.79
10	73.72	89.01	116.04	132.19	148.66	154.88	160.22	162.25
11	99.34	116.14	139.61	150.37	159.56	162.71	165.32	166.30
12	170.00	170.00	170.00	170.00	170.00	170.00	170.00	170.00

Or as a graph:







BUGS

FLOG values above 10000 or below -10000 blow out the software with a range error when you try to shoot a fade. Keep values out of this range.

Versions prior to K1.16 did not handle values between -1 and 1 correctly, i.e. did not generate a linear curve. Fixed in K2.00 (Aug 2020)

SEE ALSO

OPCS Commands

```
CAM(OPCS)             - shoot camera (fades/dissolves too)
OPN(OPCS), CLS(OPCS) - open/close fader shutter
SHU(OPCS)             - move fader to an absolute position in degrees
DXI(OPCS), DXO(OPCS) - set up dissolve in/out
FDI(OPCS), FDO(OPCS) - set up fade in/out
```

OPCSDEFS Commands

```
FLOG(OPCSDEFS)   - set Fader LOGarithmic curve for custom fades
FRANGE(OPCSDEFS) - set fade/dx's degrees range (for Hicon film stocks)
INTERP(OPCSDEFS) - set interpolation positions (fader, focus, etc)
SLOP(OPCSDEFS)   - correct for slop in a motor (fader, focus, etc)
```

General

```
MATH(DOCS)       - math expressions (for use in frame specifications)
SYNTAX(DOCS)     - online calculator and OPCS math expression syntax
```

ORIGIN

Gregory Ercolano, Los Feliz California 12/14/89

FPF(OPCSDEFS) Optical Printer Control System FPF(OPCSDEFS)

NAME

fpf - configure 'frames per foot' for a shutter motor

SYNOPSIS

fpf [chan] [frames per foot]

EXAMPLE

```
fpf a 8       # aerial projector is Vistavision (8 frms/ft)
fpf b 16      # main projector is 35mm (16 frms/ft)
fpf c 40      # camera is 16mm (40 frms/ft)
fpf c 0       # disable foot/frames counters for camera (K2.02+)
```

DESCRIPTION

Sets the number of frames in a foot of film for each axis. This value is used by the camera/projector counters to calculate for the feet/frames display, as well as how to interpret expressions such as **cam 12'0**.

[chan] is the motor channel being configured.

[frames per foot] may be any integer value that specifies the number of frames per foot of film. In K2.02 and up, this value can be '0' to disable the feet/frames counter in the display.

BUGS

In versions older than K2.02, an FPF value of 0 was unsupported and would blow out the software with a DIVISION BY ZERO error.

Also, floating point values (such as .5) are not valid arguments for the FPF command.

ORIGIN

Gregory Ercolano, Los Feliz California 11/29/89

FRANGE(OPCSDEFS) Optical Printer Control System FRANGE(OPCSDEFS)

NAME

frange - configure the open/close positions for fades/dissolves

USAGE

```
frange [close] [open]         #  values are in degrees
```

EXAMPLES

```
frange 0 170        # normal for a 170 degree shutter

frange 90 130       # for special Hicon film stocks
                    # (fades/dissolves go from 90 to 130
                    # degrees)
```

DESCRIPTION

FRANGE(OPCSDEFS) allows the user to specify the open and closed positions [in degrees] for the FADE/DISSOLVE commands.

NOTES

```
**********************************************************************
** This command does NOT affect the OPN/CLS commands, which still go **
** full open/full closed.                                            **
**********************************************************************
```

The fader must be in the correct start position for a fade/dissolve to occur without giving an error. If the FRANGE is set to 90/150, then the fader must be at 90 degrees before doing a FDI/DXI, and 150 degrees before doing a FDO/DXO.

```
**********************************************************************
** Use the 'shu' command to correctly position the fader before      **
** doing a fade/dissolve with a custom FRANGE configured.           **
**********************************************************************
```

EXAMPLE USE

By wedging the fader positions, you can find at what points light becomes exposed on the film. You can then set the FRANGE parameters to the low and high values that you find from wedging. You can then make two small scripts that turn the special fade/dissolve setup on and off. Make the following two files:

```
********************
***  0-170.RUN    ***
********************
@#  ENABLE NORMAL FADES AND DISSOLVES (0-170)
@! echo frange 0 170 >  foo.defs ! ldefs foo.defs
@! echo FADES/DISSOLVES  NOW RUN FROM 0 TO 170 DEGREES

*********************
***  90-150.RUN    ***
*********************
@#  ENABLE SPECIAL FADES/DXS (90=closed, 150=open)
@! echo frange 90 150 >  foo.defs ! ldefs  foo.defs
@! echo FADES/DISSOLVES  NOW RUN FROM 90 TO 150 DEGREES
```

You then need only to run the appropriate script before shooting:

```
    run 90-150.run        # setup new fader values
    shu 90                # (fader starts in new position)
    fdi 12 rep 12         # shoot 12 frame fadein using 90-150
    rep 200               # (fader is at 150 for rest of shoot)
    dxo 24 rep 12         # dissolve out from 150 to 90 degrees
    cls cam 120           # (closes fader to 0 for windoff)
    run 0-170.run         # Go back to normal fades/dissolves
```

BUGS

none reported.

SEE ALSO

OPCS Commands

```
CAM(OPCS)            - shoot camera (fades/dissolves too)
OPN(OPCS), CLS(OPCS) - open/close fader shutter
SHU(OPCS)            - move fader to an absolute position in degrees
DXI(OPCS), DXO(OPCS) - set up dissolve in/out
FDI(OPCS), FDO(OPCS) - set up fade in/out
```

OPCSDEFS Commands

```
FLOG(OPCSDEFS)   - set Fader LOGarithmic curve for custom fades
FRANGE(OPCSDEFS) - set fade/dx's degrees range (for Hicon film stocks)
INTERP(OPCSDEFS) - set interpolation positions (fader, focus, etc)
SLOP(OPCSDEFS)   - correct for slop in a motor (fader, focus, etc)
```

General

```
MATH(DOCS)       - math expressions (for use in frame specifications)
SYNTAX(DOCS)     - online calculator and OPCS math expression syntax
```

ORIGIN

Gregory Ercolano, Los Feliz California 04-21-92

```
HARDWARE(OPCSDEFS)      Optical Printer Control System      HARDWARE(OPCSDEFS)

    NAME
        hardware - enable/disable using printer hardware

    SYNOPSIS
        hardware [yes or no]

    DESCRIPTION
        'hardware no' will disable writing to ports on the PC, as well as
        all motor running routines that deal with the kuper card hardware.
        'hardware no' is useful only to use the printer software in a minimal
        way if the kuper card is not present.

        For normal use when the system is supposed to drive an optical printer,
        this command should be 'hardware yes'.

    BUGS
        none.

    SEE ALSO
        MOTORS(OPCS) - run script debugging command

    ORIGIN
        Gregory Ercolano, Los Feliz California 11/29/89
```

INTERP(OPCSDEFS) Optical Printer Control System INTERP(OPCSDEFS)

NAME

interp - configure channel position interpolations

SYNOPSIS

interp [chan] [master] [low] [high] [total] [samples]

EXAMPLES

interp d - 0 170 18 0 1500 2500 3200 3900 4500 5080 5620 6130
6670 7120 7630 8110 8550 9010 9470 9930 10340

interp F E 0 12000 13 0 10 20 30 40 50 60 70 80 90 100 110 120
|
Note that 'e' channel the master zoom, and 'f'
is the follow focus.

interp d - 0 0 0 # Disable any previous interp for 'd'

DESCRIPTION

INTERP allows a channel to be interpolated into a sampled curve, or to slave to another channel using interpolation into a sampled curve.

[chan] is the channel that is going to have the interpolation defined to it.

[master] is channel name of a master channel, such as in a zoom/follow focus relationship where the zoom is the master to the follow focus channel. In such a case, the position of the focus channel is a function of the position of the zoom channel. If mastering is not desired (such as with the fader), specify '-'.

[low] [high] These indicate the low and high range of the 'requested' positions that will be asked of this channel. These can be floating point values.

[total] This indicates how many sample points make up the interpolation curve.

NOTE: If [total] is '0', this will cancel any previous INTERP(OPCSDEFS) specifications for this channel.

[samples] are the sample positions. Start with the [low] samples, and work up towards the [high] samples. Each sample position should be separated by white space (tabs, spaces, CRLFs) and there should be as many samples as specified by [total].

OVERVIEW

When an INTERP command is set up on a channel, it is like defining a look up table through which each position request is converted to a new position, according to the value found in the lookup table.

Requests that fall between values in the lookup table are computed as a linear interpolation between the two lookup values. When enough points are supplied to the lookup table, very complex functions can be defined and approximated (to the point where error is negligible). Follow focus curves can be defined quite well with just a few dozen sample positions. Nonlinear mechanics can be compensated for, such as faders that have built in logarithmic motion, so the computer can control it in a linear fashion (such as for cross dissolves).

HOW INTERP WORKS

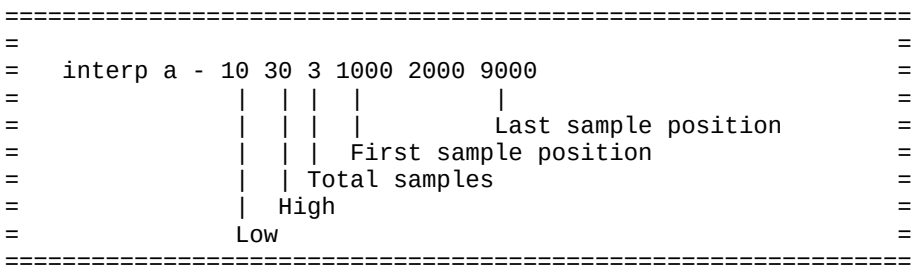


Figure A.

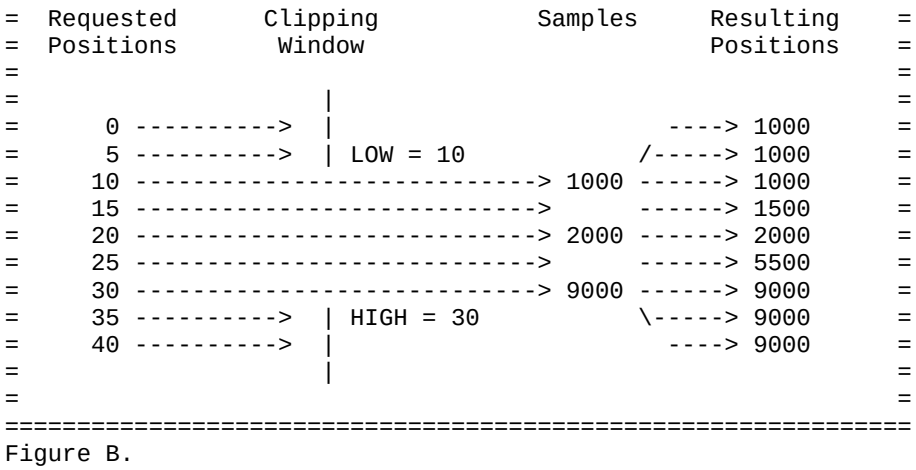


Figure B.

The 'Requested Positions' are the positions the operator wants the motors to move to. The 'Resulting Positions' are the values that are actually sought by the motors. What happens in between is caused by the INTERP command (Figure A).

Requested positions are clipped into the range LOW and HIGH. Values less than LOW are made LOW. Values above HIGH are made HIGH.

Once clipped, the values are stretch-fitted according to the samples supplied. Note how linear interpolation is used to compute the lookup for '15', which falls between the two samples 1000 and 2000, translating to 1500.

FADER INTERPOLATIONS

The software can be set up to account for weird (nonlinear) hardware such as in a fader. A sample point is supplied for every 10 degrees on the fader. The samples represent the actual motor positions for every 10 degrees in the fader. The following is an actual example for a 120 degree fader:

```
interp  D - 0 120 13 0 -260 -365 -450 -515 -585 -645
                       -705  -760  -815  -860  -910  -970
```

Note how the LOW and HIGH values are set to 0 and 120. This way a request for 120 will actually move the fader to its open position which is the actual position of -970, and 0 will move the fader to its actual position of zero.

The samples are nonlinear, and represent the number of steps for every 10 degrees in the fader. These points were found by first disabling any INTERPs on the fader, and then moving the motor several pulses at a time, taking note of the step position every time the fader hit a 10 degree mark.

FADER SETUP

Assuming you have a fader with some sort of logarithmic mechanical rig, you need to follow this procedure to set up your fader properly to counteract the built in logarithmic movement (for proper dissolves).

- Make sure there is no INTERP command already set up on the fader in the OPCSDEFS.OPC file. If there is, comment it out with '#', and rerun the software, or disable it by typing the following:

```
ldefs  con
interp d - 0 0 0      # cancels interpolations
^Z                    # (control-Z and return)
```

- Check for any significant mechanical slop in the fader:

  Using the JOG command, move the motor forward. Now change directions, stepping the motor a small amount at a time. Make note of how many pulses it takes before the fader actually starts moving in the other direction. If it begins moving immediately, there is no need to set up slop correction for the fader. If there is a great deal of slop, refer to SLOP(OPCSDEFS) to setup slop correction before continuing with this procedure.

- Before continuing, you will need to have an accurate idea of the degree positions of the fader. For truly accurate measurement, you should remove the front of your camera, and using a protractor, put scribe marks for every 10 degrees along the outer edge of the fader's shutter. Using a scribe on the camera body as a pointer, you should be able to find each 10 degree position accurately.

- Use JOG(OPCS) to position the fader to the CLOSED position. Move in small steps until you get the fader properly closed, which usually involves overlapping the fader/shutter blades a little. Avoid changing direction when finding the position to minimize slop errors.

- Now, reset the fader's counter to zero. You can do this from within JOG, or by executing:

```
reset d 0
```

- Now, slowly JOG the fader shutter to each 10 degree mark (10,20..). Each time the fader hits a 10 degree mark, write down the motor position from the display.

  Avoid going too far, and especially avoid changing direction. If you screw up by going too far, start over from the beginning with the shutter CLOSED.

- If you have a 170 degree fader, you will have 18 values (including '0' for the 0 degrees position). Using a text editor, make an INTERP command for the fader that contains all the samples you just found:

```
interp d - 0 170 18 0 112 153 278 475  ...
             |   |  ----------------------
             |   |           |
             |   |    The samples you found. (include 0)
             |   |
             |   Total samples you found. (including 0)
             |
            The number of degrees in your fader.
```

If there are more samples then can fit on a line, you can let them wrap around the screen, or embed carriage returns and tabs for readability.

* * *

You should now be able to start up the OPCS software, and position the motor to correct positions with the SHU(OPCS) command, i.e. **shu 50** should send the shutter to the 50 degree position.

Dissolves and fades should also work properly now.

If your fader mechanics suffers from slop, e.g. if the fader slips when it changes direction, refer to the SLOP(OPCSDEFS) command for ways to overcome mechanical slop.

BUGS

If you have very few pulses between an OPEN and CLOSED shutter, you may experience 'rounding' problems in the display, i.e. using shu 50.5 may actually send the shutter to 50.1 or some such deviation because your hardware may not have enough resolution to actually achieve 50.5.

SEE ALSO

SHOW(OPCS) - show current positions for all motors
JOG(OPCS) - interactively jog a motor to positions
GO(OPCS) - move motors some distance or to new positions
SHU(OPCS) - send fader to absolute positions in degrees
SLOP(OPCSDEFS) - overcome mechanical slop, esp. faders

ORIGIN

Gregory Ercolano, Los Feliz California 12/20/90

```
JOGSTEP(OPCSDEFS)       Optical Printer Control System      JOGSTEP(OPCSDEFS)

    NAME
        jogstep - set the jog mode's 'vernier' and 'crawl' step rate

    USAGE
        jogstep [vsteps] [csteps]

        where:
            [vsteps] is the number of steps per keypress in vernier modes.
            [csteps] is the number of steps per keypress in crawl modes.

    EXAMPLES
        jogstep 5 50    # recommended for microstepper systems
        jogstep 1 10    # recommended for half stepper systems

    DESCRIPTION
        JOGSTEP(OPCSDEFS) lets you tailor the number of steps moved per
        keypress when in the jog mode. Higher values make the motor move more
        per keypress, smaller values make movement more accurate (and slow).

    SEE ALSO
        JOG(OPCS) - jog motors interactively

    ORIGIN
        Gregory Ercolano, Los Feliz California 08/25/91
```

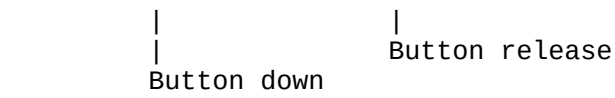
KEYFUNC(OPCSDEFS) Optical Printer Control System KEYFUNC(OPCSDEFS)

NAME

keyfunc - lets user define keys in KEY(OPCS) and JOG(OPCS)

USAGE

keyfunc -clear "function"
keyfunc -add "function" port mask test port mask test



"function" is either the name of an internal function (see below) or an opcs command string. In the case of -clear, "function" can be "all", to clear all previously defined functions.

When binding a port/bit mask to an OPCS function or OPCS command string, one specifies the port/bits for both 'button down' and 'button release'.

DESCRIPTION

This command allows the user to define the keyboard keys used in KEY(OPCS) mode. The user can actually assign the operations to not only any keys on the keyboard, but any bit on any port on the IBM PC.

This allows for external buttons to control bits on the parallel port (or whatever ports are available) and thus control any of the functions supported by the KEY(OPCS) command.

A maximum of 200 keyboard functions can be defined with 'keyfunc'.

The following is a list of all functions the KEY(OPCS) command currently supports, plus some custom definitions. Normally, these commands would appear in the OPCSDEFS.OPC setup file:

The following shows all the built-in functions, and show examples of how to define custom commands.

```
# CLEAR ALL FUNCTIONS FIRST
#     Start with a completely clean slate.
#
keyfunc -clear "all"

# DEFINE ALL THE 'BUILTIN' OPCS FUNCTIONS
#     These names are the names of built in operations in OPCS,
#     whose operation should be obvious. These are all assigned
#     to keyboard scan codes. Comments at right describe what the
#     scan codes are.
#
#            OPCS FUNCTION    KEY DOWN      KEY RELEASE
#            -------------    ----------    -----------
keyfunc -add "quit"           0060 ffff 01  0060 80 80  # ESC
keyfunc -add "pro2fwdslew"    0060 ffff 3b  0060 80 80  # F1
keyfunc -add "pro2revslew"    0060 ffff 3c  0060 80 80  # F2
keyfunc -add "pro2fwd1"       0060 ffff 3d  0060 80 80  # F3
keyfunc -add "pro2rev1"       0060 ffff 3e  0060 80 80  # F4
keyfunc -add "pro1fwdslew"    0060 ffff 3f  0060 80 80  # F5
keyfunc -add "pro1revslew"    0060 ffff 40  0060 80 80  # F6
keyfunc -add "pro1fwd1"       0060 ffff 41  0060 80 80  # F7
keyfunc -add "pro1rev1"       0060 ffff 42  0060 80 80  # F8
keyfunc -add "camfwdslew"     0060 ffff 43  0060 80 80  # F9
keyfunc -add "camrevslew"     0060 ffff 44  0060 80 80  # F10
keyfunc -add "camfwd1"        0060 ffff 57  0060 80 80  # F11
keyfunc -add "camrev1"        0060 ffff 58  0060 80 80  # F12
keyfunc -add "rep+1"          0060 ffff 02  0060 80 80  # 1
keyfunc -add "rep-1"          0060 ffff 03  0060 80 80  # 2
keyfunc -add "repset"         0060 ffff 04  0060 80 80  # 3
keyfunc -add "pro2set"        0060 ffff 05  0060 80 80  # 4
keyfunc -add "pro1set"        0060 ffff 06  0060 80 80  # 5
keyfunc -add "camset"         0060 ffff 07  0060 80 80  # 6
keyfunc -add "fdiset"         0060 ffff 08  0060 80 80  # 7
keyfunc -add "fdoset"         0060 ffff 09  0060 80 80  # 8
keyfunc -add "dxiset"         0060 ffff 0a  0060 80 80  # 9
keyfunc -add "dxoset"         0060 ffff 0b  0060 80 80  # 0
keyfunc -add "cls"            0060 ffff 0c  0060 80 80  # -
keyfunc -add "opn"            0060 ffff 0d  0060 80 80  # =
keyfunc -add "seek"           0060 ffff 0e  0060 80 80  # BS

# DEFINE SOME CUSTOM OPCS COMMANDS
#     These are not builtin functions, but are simply run as opcs commands.
#     You can run OPCS scripts, and invoke DOS commands (if prefaced with "!")
#
keyfunc -add "home"             0060 ffff 004e  0060 0080 0080   # Num +
keyfunc -add "load"             0060 ffff 0037  0060 0080 0080   # Num *
keyfunc -add "run myscript.run" 0060 ffff 004a  0060 0080 0080   # Num -
```

MONITORING THE KEYBOARD

Since the keyboard is mapped to port 0060, any key on the keyboard can be bound to an OPCS function, e.g.

```
    keyfunc "function_name"  0060 <mask> <test>  0060 <mask> <test>
                             ------------------  ------------------
                             Detect Key Press    Detect Key Release
```

The <port>, <mask>, and <test> values are all specified in hexadecimal.

(New in K2.20TC): To support extended keycodes, the <mask> and <test> values can now be 16 bit (4 digit hex) values. OPCS now treats the keyboard (port 0060) as a special case, where extended keycodes have the high byte set to E0 (e.g. E0##), and regular keycodes have the high byte set to 00 (e.g. 00##) for "normal" keys.

To detect a KEY DOWN, use FFFF as the 'mask', and the 16bit scan code value as the 'test' value. Single byte scancodes will be 00##, extended scancodes will be E0##.

To test for KEY RELEASE, use 0080 as both 'mask' and 'test' value, since all key release codes have the 0080h bit set.

So for example, to program the "F1" key which generates 003B on press and 00BB on release:

```
  #            OPCS FUNCTION   PORT MASK TEST  PORT MASK TEST
  #            --------------  ---- ---- ----  ---- ---- ----
  keyfunc -add "pro2fwdslew"   0060 ffff 003b  0060 0080 0080
```

..and to detect e.g. the "PrntScrn" key which generates E0 4A when a key is pressed, and E0 CA on release, this works:

```
  #            OPCS FUNCTION   PORT MASK TEST  PORT MASK TEST
  #            --------------  ---- ---- ----  ---- ---- ----
  keyfunc -add "load"          0060 ffff e04a  0060 0080 0080
                                       \__/
                                  16bit extended scancode
                                  with E0 in high byte
```

GENERAL PORT MONITORING

A single function, such as "pro1fwdslew", can be assigned to more than one port/bit combination, allowing several ways to access the same function. For example, you may want to have both keyboard and external buttons to slew the projector forward.

For a list of all the keyboard scan codes, refer to an IBM PC technical manual, or run "! key" from the DOS prompt to run the KEY.EXE program, which prints the hexadecimal scancodes for any key you hit.
Hit ESC to exit the KEY.EXE program.

CONTINUOUS SWITCH

The 'BUTTON RELEASE' port/mask/test definitions define the IBM PC port to monitor to determine when the key is released. This can optionally be defined to monitor a 'continuous' switch, so that the motors will stop when the continuous switch is released:

keyfunc -add "profwd1" 03bd 40 40 03bd 41 00

The above monitors port 0x03bd for the 0x40 bit. When set, the motor will run until the 0x40 and the 0x01 bit of the same port are zero. In this case, the 0x40 bit is probably the invoking button, and the 0x01 bit is the 'continuous' switch; if on, the motor will run until the continuous switch is turned off, even if the 0x40 bit is released.

BUGS/CAVEATS

If you use **keyfunc -clear "all"**, you must at LEAST declare the quit key. If you don't, you will have no way to break out of the KEY(OPCS) or JOG(OPCS) modes.

When using keyfunc to invoke an OPCS command string, the command should not exceed 256 characters. If you want a single keystroke to invoke many commands, make a script, then have the keyfunc definition run the script, e.g.:

keyfunc -add "run fancy.run" 0379 80 80 0379 80 00

SEE ALSO

KEY(OPCS) - use keys to run motors
JOG(OPCS) - jog motors interactively
DEENERGIZE(OPCSDEFS) - define port/bit to deenergize motors
ALLSTOP(OPDSDEFS) - define port/bit to detect the allstop key
BUCKLE(OPCSDEFS) - define port/bit to detect film buckles
VIEWER(OPCSDEFS) - define port/bit to detect viewer open
TRIPSWITCH(OPCSDEFS) - define port/bit to detect trip switches
SETBIT(OPCSDEFS) - set bit(s) on a port
CLRBIT(OPCSDEFS) - clear bit(s) on a port
XORBIT(OPCSDEFS) - invert bit(s) on a port

ORIGIN

Gregory Ercolano, Venice California 04/18/98

LOGCOUNTERS(OPCS) Optical Printer Control System LOGCOUNTERS(OPCS)

NAME

logcounters - configure if cam/pro counters should be logged

SYNOPSIS

logcounters [on/off]

EXAMPLES

```
logcounters on      # log files will contain counter info
logcounters off     # DON'T put counter info into log files
```

DESCRIPTION

When LOGCOUNTERS is on, and a LOG(OPCS) command is logging out to a file or line printer, the motor positions are logged along with each line the operator types. The positions are entered into the file as comments, so when executed, the position information is ignored.

The format of how the counters are logged is configured with LOGFORMAT(OPCSDEFS).

This information may be necessary for debugging purposes, but in most cases just 'clutters up' an otherwise easy to read file, and it may be desirable to leave this setting off when logging to files.

BUGS

If the operator uses ALLSTOP during command logging, it would be unwise to later execute the log file as a run script without making the proper modifications to the commands that were interrupted. Here is a sample log with a command that was interrupted:

```
#  1:1   0(0'0)         20(1'4)       CLOSED 0
cam -120

# ### OPERATOR HIT ALLSTOP KEY
#  1:1   0(0'0)         18(1'2)       CLOSED 0
```

Note the camera counter now reads 18 instead of -100. Because the command was interrupted, it never got to finish shooting. This could cause confusion later if this log were executed as a RUN script, and commands that followed used absolute positioning (cam >134). The command **cam -120** should then modified by hand:

cam >18 or **cam -2**

...to reflect the command as it was actually executed.

SEE ALSO

```
LOG(OPCS)               - log all commands entered by the user
RUN(OPCS)               - run a log file
LOGCOUNTERS(OPCSDEFS)   - enable/disable logging counters to logfiles
LOGFORMAT(OPCSDEFS)     - formats how values are printed to logfile
```

ORIGIN

Gregory Ercolano, Los Feliz California 12/15/89

```
LOGFORMAT(OPCSDEFS)   Optical Printer Control System    LOGFORMAT(OPCSDEFS)
```

NAME

logformat - format how counters appear in log files

SYNOPSIS

logformat string

EXAMPLES

logformat \# %0ar:%0br:%0cr %13aF %13bF %13cF %0dS\n
logformat \# Camera Count=%13cF, Command:

An example output using the above format string might yield the following in the log file:

```
        # 0:1:1   26(1'10)      0(0'0)      0(0'0)  170.00
          -----   --------      ------      ------  ------
            |       |             |           |       |
            |       |             |           |       |
          Ratio   Aerial        Main        Camera  Fader
                  Counter       Counter     Counter Position
```

DESCRIPTION

When the camera operator enables LOG(OPCS) and LOGCOUNTERS(OPCSDEFS) is enabled, LOGFORMAT(OPCSDEFS) sets the format string used to determine how the counters are printed in the logs.

The format string can contain 'format sequences' which are replaced with various dynamic counter values when printed to the log. Like printf() in the C language, backslash escape sequences are honored:

```
        \r  - carriage return (no line feed)
        \e  - escape
        \n  - a carriage return/line feed
        \t  - tab
```

..and OPCS-specific '%' format sequences, such as:

```
        %5cF     -- print channel 'c' counter in frames(ft'frms) format
         |||        (e.g. 16(1'0) with 5 digit padding
         |||
         ||Indicates "frames(feet)" format
         |Channel 'c' (camera)
          The number of digits to pad
```

..which is an example of the general syntax:

```
        %<width><channel><op>
```

..where:

<width> a numeric value of how many characters to pad.

<channel> the channel letter for the counter. ('a'-'p')

<op> the format operator character which can be one of:

r - ratio value, e.g. "1:1:1"
p - position, e.g. "26"
f - feet'frames, e.g. "1'10"
F - frames(feet), e.g. "26(1'10)"
S - special (currently, "%0dS" prints fader in degrees)

'%%' can be used to print a percent character. (OPCS K1.13b+)
Any other characters are inserted in the logfile verbatim.

(OPCS K2.03) If FPF(OPCSDEFS) is set to 0 for a channel,
"-" will be printed in place of a feet'frames value.

BUGS

Verisions of OPCS earlier than K2.03 would blow out with a
DIVISION BY ZERO error if FPF(OPCSDEFS) is set to zero for a channel.

In K2.03, an fpf of zero is supported to disable footage counts,
such as for film stocks that have a non-integer number of
frames per foot, e.g. IMAX 15 perf (4.26666) and 10 perf (6.40).

ORIGIN

Gregory Ercolano, Venice California 04/07/98

NAME

mrp - maximum ramp pulses for shutter motors

USAGE

mrp [chan] [pulses]

EXAMPLES

```
    mrp b 800     # limit pro ramps to no more than 800 pulses
    mrp c 500     # limit cam ramps to no more than 500 pulses
                  # during shutter runs
```

DESCRIPTION

Sets a maximum for the number pulses used for ramping during shutter runs for the specified channel. Set this value equal to or less than the number of pulses it takes to move the exposing shutter from the 'closed' position to the position where the shutter just begins to open.

This will ensure motor ramping does not occur while the film is being exposed.

Use the following logic:

> If there are 2000 steps per full rotation of the shutter, and the shutter starts opening at 90 degrees (1/4 rotation) on either side of the closed position, ramping can therefore occur up to 500 pulses without risking unintended film exposure.

In actual practice, you may want to set the value slightly lower, in case the shutter has slop.

Only channels with a shutter blade (camera, cap shutter) need an MRP value; the rest can be 0.

To accurately calculate the MRP value for the camera, one must understand exactly at what point film exposure takes place, e.g.

Shutter Exposure	Degrees	5000 PPR	2000 PPR	
Full Close	0	0	0	__
First Light	76	1056	422	| Film Exposed to light
Full Open	112	1556	622	|__ during this period.
First Dark	245	3403	1361	| Avoid motor ramping
Full Dark	283	3931	1572	__| during this time!
Full Close	360	5000	2000	

So for a 2000 PPR shutter, we want to limit motor ramping to the 0..400 and 1600..2000 region, to avoid ramping during exposure. Therefore the MRP should be 400, which limits rampup to the 0..400 region, and rampdown to the 1600..2000 region.

```
    In actual practice there's slop in both the home and shutter
    movement, but also the "First Light" and "Full Dark" are not
    very critical, as very little exposure is happening during
    this time, so often MRP 400 can safely be 500 (or 90 deg).

    For camera MRP, it's the degree positions of the shutter blade
    which determines when light can first start hitting the film
    through the Acme 35mm full frame aperture. The formula for
    degrees(out of 360) to pulses (out of the PPR or Pulses Per Rev):

             pulses = ( <deg> / 360 ) * <PPR>
             1056   = (  76   / 360 ) * 5000    <- for a 5000 PPR shutter
              422   = (  76   / 360 ) * 2000    <- for a 2000 PPR shutter

 CAVEATS
    MRP also affects the operation of AUTOFILT(OPCS) and FILTER(OPCSDEFS).
    See these documents for details.

SEE ALSO
    MRP(OPCSDEFS)     - set 'maximum ramping pulses' for shutter runs
    RAMP(OPCSDEFS)    - set maximum accelerations and velocities
    SPD(OPCS)         - set the camera's exposure speed
    SPD(OPCSDEFS)     - set a motor's running speeds
    RAMPCURVE(OPCSDEFS) - set ramping curves for shutter runs
    AUTOFILT(OPCS)    - enable/disable the auto-wedging filter wheel
    FILTER(OPCSDEFS)  - define channel to control a filter wheel

ORIGIN
    Gregory Ercolano, Los Feliz California 08-15-91
```

```
NAME(OPCSDEFS)          Optical Printer Control System          NAME(OPCSDEFS)

    NAME
        name - set the channel's name used in counter displays

    SYNOPSIS
        name [chan] name

    EXAMPLES
        # Use names that match the commands
        name a Pro2  # sets name for 'a' channel to "Pro2"
        name b Pro1  # sets name for 'b' channel to "Pro1"
        name c Cam   # sets name for 'c' channel to "Cam"

        # Use names that match the channel letter
        name a A     # sets name for 'a' channel to "A"
        name b B     # sets name for 'b' channel to "B"
        name c C     # sets name for 'c' channel to "C"

    DESCRIPTION
        Lets the operator set the default names used in the counter
        displays, e.g. bigcounters(OPCSDEFS), show(OPCS), etc.
        The 'name' parameter:

             > Must not contain spaces
             > Are limited to 9 characters in length

        If spaces are needed, use underbars instead (_).
        'name' is limited to 9 characters per channel due to the
        screen constraints of 80 column displays for onscreen counters.

    HISTORY
        This command was added in OPCS version K2.00/TC (Turbo C).
        Older versions (K1.xx) used STARTUP.DEFS which was obsoleted
        in K2.00, and in many cases hard-coded names were used instead.

    SEE ALSO
        BIGCOUNTERS(OPCSDEFS) - sets the style of the onscreen counters

    ORIGIN
        Gregory Ercolano, Alhambra California 07/16/21
```

```
OPCSCMD(OPCSDEFS)       Optical Printer Control System      OPCSCMD(OPCSDEFS)

    NAME
        opcscmd - execute an OPCS command from within a defs file

    USAGE
        opcscmd [command command command]

    EXAMPLES
        opcscmd go d 50 reset d 0      # move d chan, then reset to 0

    DESCRIPTION
        OPCSCMD(OPCSDEFS) allows a gateway from the defs file commands to the
        OPCS commands, the same way LDEFS(OPCS) allows OPCSDEFS commands to
        be executed from within the OPCS command mode.

        All text that follows 'opcscmd' up to the end of the line (or a '#'
        comment character) is executed as an OPCS command string.

    SEE ALSO
       LDEFS(OPCS)      - run OPCSDEFS commands from within the OPCS command mode
       OPCSCMD(OPCS)    - run OPCS commands from within the OPCSDEFS command mode
       man -k OPCS:     - list OPCS commands with 'one liner' descriptions
       man -k OPCSDEFS: - list OPCSDEFS commands with 'one liner' descriptions

    ORIGIN
        Gregory Ercolano, Los Feliz California 12/15/89
```

PPR(OPCSDEFS) Optical Printer Control System PPR(OPCSDEFS)

NAME

ppr - configure the 'pulses per revolution' for a motor

SYNOPSIS

ppr [chan] [pulses]

EXAMPLE

```
ppr a 2000            # microstepper system
ppr a 4000            # microstepper with vistavision
ppr a 400             # half stepper system
```

DESCRIPTION

Sets the number of pulses needed to rotate a motor one revolution. This command exists especially for the CAMERA and PROJECTOR motors.

Keep in mind the OPCS hardware runs stepper motors at more than the motor's rated resolution. Microstepper systems can have as many as 2000 pulses per rev, and half stepper systems can have 400 per rev.

The software uses the PPR value in two ways. One is to obviously translate frames into physical steps for the motors. The other is for the ALLSTOP logic to know when to check for the ALLSTOP key, so as not to stop the shutter mid-revolution.

NOTES

PPR settings for the fader are never used by the software, since revolutions have no meaning in the context of running the shutter.

PPR values should be divisible by two, esp. for the projector so that half phase shifts calculate to non-fractional steps.

BUGS

You cannot specify floating point values for PPR. This is not actually a bug: if your hardware is geared in such a way that a full revolution occurs in a fractional number of steps, you should probably fire the guy who built it and have the hardware rebuilt anyway.

To avoid a nasty bug with the ALLSTOP key, and to have counters update properly, configure the PPR(OPCSDEFS) command for the D thru L channels to be '10' in your OPCSDEFS.OPC file, regardless of the actual number of pulses per revolution. This also ensures that slewing in JOG doesn’t go in very large increments.

```
ppr  d  10  # non-shutter channels only
ppr  e  10
ppr  f  10
ppr  g  10
ppr  h  10
```

ORIGIN

Gregory Ercolano, Los Feliz California 11/29/89

```
PRO2DISPLAY(OPCSDEFS)  Optical Printer Control System  PRO2DISPLAY(OPCSDEFS)

    NAME
        pro2display - Enable/Disable display of aerial projector counters

    USAGE
        pro2display [on|off]

    EXAMPLES
        pro2display on     # show aerial projector counters
        pro2display off    # don't show aerial projector counters

    DESCRIPTION
        Some printer systems do not have aerial projectors. This command
        disables the aerial counters, to avoid confusing camera operators.

    ORIGIN
        K1.12d+ Gregory Ercolano, Venice California 04/09/98
```

```
PROPHASE(OPCSDEFS)      Optical Printer Control System    PROPHASE(OPCSDEFS)

    NAME
        prophase - configure projector's phase adjustment for 1:1 shooting

    USAGE
        prophase [chan] [pulses]   # sets number of pulses channel
                                   # will move before shooting 1:1

    EXAMPLES
        prophase a 200             # half-stepper systems (400 ppr)
        prophase b 200             #

        prophase a 1000            # Centent microsteppers (2000 ppr)
        prophase b 1000            #

        prophase a 1600            # Lynx microsteppers (3200 ppr)
        prophase b 1600            #

    DESCRIPTION
        Sets the projector phase adjustment (in motor pulses). Phase adjustment
        occurs just before and just after 1:1 ratio shooting. This phase
        adjustment is necessary so the projector(s) and camera can run
        together, only exposing film when the projector images are seated.

        Almost without exception, the prophase command:

                1) Is always set to be 1/2 the number of pulses for  one
                   motor revolution. This includes  Vistavision movements.
                   On a system where one switches back and forth  between
                   Vista / 35mm / 16mm, the prophase value does NOT  change,
                   even though the PPR(OPCSDEFS) value may  change.

                2) Is set ONLY for the projectors. A value of '0' should  be
                   set for all the other  channels.

                   The prophase value is actually ignored by the other
                   channels, so it doesn't really matter what the values
                   are for non-projector channels. But specifying '0'
                   makes it clear the value is ignored.

    BUGS
        None reported.

    SEE ALSO
        RAT(OPCS)           - set the shooting ratio for the REP command
        REP(OPCS)           - shoot current projector/camera shooting ratio
        PROPHASE(OPCSDEFS)  - sets projector phase adjustment for 1:1 shooting
        MATH(DOCS)          - math expressions (for use in frame specifications)
        SYNTAX(OPCS)        - Online calculator and OPCS math expression syntax

    ORIGIN
        Gregory Ercolano, Los Feliz California 12/15/89
```

RAMP(OPCSDEFS) Optical Printer Control System RAMP(OPCSDEFS)

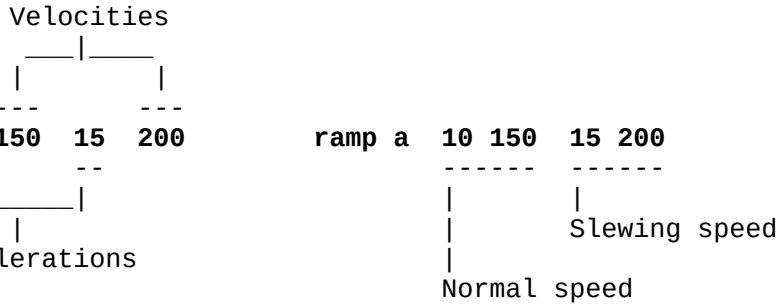
NAME

ramp - configure motor's maximum accelerations and velocities

USAGE

ramp [chan] [norm accel] [norm vel] [slew accel] [slew vel]

EXAMPLES



DESCRIPTION

This command sets up the maximum accelerations and velocities for non-shutter motor movement commands such as GO(OPCS), JOG(OPCS), FEED(OPCS), etc.

Note: RAMP(OPCSDEFS) is not used for shutters and film movement commands, e.g. CAM(OPCS), REP(OPCS), SEEK(OPCS). Those calculate accelerations and velocities automatically based on motor speeds configured with SPD(OPCS) and SPD(OPCSDEFS), and are influenced by PPR(OPCSDEFS) and MRP(OPCSDEFS).

Since the kuper card works at 120 samples per second, the values represent the number of pulses per sample, or the number of pulses per 1/120th of a second.

Acceleration values are used for ramping. As the motor comes up to speed, it will do so in increments specified by the 'acceleration'.

Velocity values are the maximum running velocities. The motor will not go faster than the maximum velocity supplied.

SEE ALSO

MRP(OPCSDEFS) - set 'maximum ramping pulses' for shutter runs
RAMP(OPCSDEFS) - set maximum accelerations and velocities
SPD(OPCS) - set the camera's exposure speed
SPD(OPCSDEFS) - set a motor's running speeds
RAMPCURVE(OPCSDEFS) - set ramping curves for shutter runs

ORIGIN

Gregory Ercolano, Los Feliz California 08-15-91

RAMPCURVE(OPCSDEFS) Optical Printer Control System RAMPCURVE(OPCSDEFS)

NAME

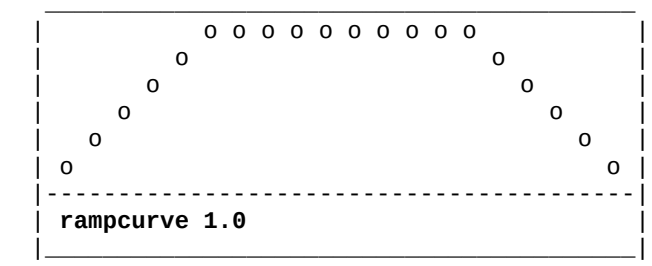
rampcurve - configure the motor ramping curve for shutter runs

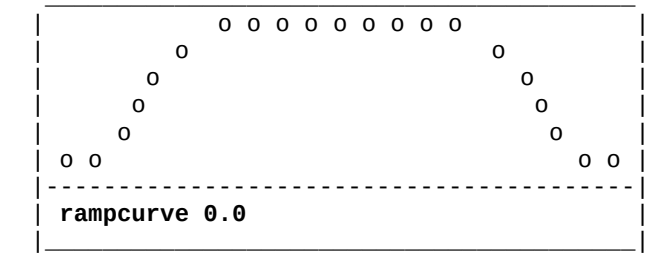
USAGE

```
rampcurve [value]              # 1.0 = linear, 0.0 = sinusoidal
```

[value] should be in the range -0.3 to 1.0. Values outside this range are not recommended.

EXAMPLES





<-- recommended setting

DESCRIPTION

This command allows the user to define the curve used during ramping for the shutter motors. The effects are subtle at higher speeds.

The linear graph is desirable for motors that are moving a great deal of torque, or overcoming a large inertia.

The sinusoidal graph (if you can tell which that is, what with our 'high resolution' display capabilities here) is for motors that can quickly get up to speed, without torque problems.

The sinusoidal ramp is preferable, in order to avoid motor resonance at the slower speeds. (Normally, all stepper motors resonate more at slower speeds, and this can be counterproductive during ramping.)

SEE ALSO

```
MRP(OPCSDEFS)     - set 'maximum ramping pulses' for shutter runs
RAMP(OPCSDEFS)    - set maximum accelerations and velocities
SPD(OPCS)         - set the camera's exposure speed
SPD(OPCSDEFS)     - set a motor's running speeds
RAMPCURVE(OPCSDEFS) - set ramping curves for shutter runs
```

ORIGIN

Gregory Ercolano, Los Feliz California 12/15/89

RESPOND(OPCSDEFS) Optical Printer Control System RESPOND(OPCSDEFS)

NAME

respond - configure device name to send responses to OPCS commands

SYNOPSIS

respond off # no com port error logging (default)
respond com1 # send error codes to com1

DESCRIPTION

This command enables error code characters to be sent to a device whenever the OPCS system is ready for a new command. The name can be any DOS device name, or can be 'off' to disable the transmission of responses completely.

Since the printer software can be started with its input coming from a device (such as: opcs < com1), RESPOND(OPCSDEFS) is used to close the loop by sending signals back to the device whenever an OPCS command completes execution to indicate when a command (or series of commands) finished executing, and whether the command failed execution or not.

Here is how to start up the printer software reading the serial port for incoming commands from a remote computer:

opcs <com1

Assuming baud rates have already been set up with the DOS 'mode' command, and the serial cable configured to properly handshake with the IBM PC (see below), the software will receive OPCS commands the same way they would be expected from the keyboard.

With 'respond' enabled, error codes will get sent back to the remote computer over the serial line to indicate when the OPCS software completed the last command and is ready for a new one.

CAVEATS

In version K2.01 and up, values other than 'off' overrides the CMDLINE(OPCSDEFS) 'editor' setting, forcing it to 'dos' mode editing. This is because interactive editing is not supported over the com port.

In the version before K2.01, 'respond' did not do the override, so you have to set 'cmdline dos' for com port communication to work.

ERROR CODES

The error protocol is pretty simple. With RESPOND(OPCSDEFS) enabled, these ASCII codes are sent back to the remote computer whenever the OPCS software is ready for a new command. Which character gets sent depends on whether the last command executed successfully or not:

CODE	DESCRIPTION
>	Command completed OK, waiting for a new command.
X	Command failed with an error, waiting for a new command.

IMPLEMENTING REMOTE COMPUTER CONTROL

When setting up another computer to control the software through the serial port, consider these issues:

1) BAUD RATE. The baud rate should be low (300 or 1200 baud) because the IBM PC does not normally do interrupt driven communications without a special driver loaded.

2) SERIAL PORT WIRING. The PC is picky about having certain serial control signals before it can communicate to other computers.

3) GETTING THE MACHINES COMMUNICATING. First, get the remote computer sending characters to the OPCS system's IBM PC, then work on getting characters back to the remote:

   a) Set up the remote to communicate at 300 baud, no parity, 8 data bits, one stop bit.

   b) Run the following on the OPCS system's computer:

      mode com1:300,n,8,1

   c) Now execute the following to test for receiving lines from the remote computer:

      type com1

      If the command fails with a timeout error, make sure pins 5 & 6 are being pulled high on the PC's 25 pin serial connector (6 & 8 on a 9 pin connector).

      If the remote computer is not pulling these signals properly, you can cheat the signals high by doing the following at the PC's connector:

25 PIN CONNECTOR	9 PIN CONNECTOR
Tie pin 20 to pin 6.	Tie pin 4 to pin 6.
Tie pin 4 to pin 5.	Tie pin 7 to pin 8.

   d) Now try sending characters to the remote:

      echo test > com1

      This should send 'test' and a CR/LF to the remote computer.

With these steps complete, the following command will force the OPCS software to receive its commands from the remote computer (you may want to have already setup a 'respond com1' command in the 'OPCSDEFS.OPC' file):

opcs < com1

Remember the the IBM PC likes to see a CR followed by a LF at the end of each line.

4) HANDSHAKING. The remote should always wait until one of the error codes has been received before sending a new command. If the returned code shows an error condition, it should probably stop sending commands, and print an error locally, since a film buckle may have caused the error, and should not continue shooting until the error has been corrected.

ALLSTOP

The ALLSTOP(OPCSDEFS) command should probably monitor the appropriate COM port address so that the remote computer can stop the motors if it wants to. The best approach is to monitor the com port's DTR bit for the port. This way the remote computer can pull the DTR line to stop the motors. The remote would simply pull the DTR signal until an error code is returned, indicating the motors have stopped.

With RESPOND(OPCSDEFS) active, all error handlers default to 'abort' so that the remote system can assume even after an error, the software is still expecting OPCS commands, and not error recovery keypresses.

ORIGIN

Gregory Ercolano, Los Feliz California 02-08-91

RUNCMD(OPCSDEFS) Optical Printer Control System RUNCMD(OPCSDEFS)

NAME

runcmd - define your own OPCS command as a RUN script

USAGE

```
runcmd [name] [filename] [#args] # create new (or redefine old) runcmd
runcmd -clear                    # clears all runcmd definitions (K2.02+)
```

[name] The name of the new command. 10 characters max.

[filename] The full pathname of the script to be executed in place of [name] when executed as a command by the operator.

NOTE: If [filename] is a '-' (dash), this will delete any commands previously defined as [name]. (ie. 'runcmd w - 0' will delete any previously defined 'w' command). [filename] is 80 characters max.

[#args] is the number of arguments OPCS will pass to the script. NOTE: If -1 is specified, the arguments can be variable; all arguments the user specifies up to the end of the line are passed on to the script.

See below 'ARGUMENTS' for how to pass arguments to a script. Any value 0-9 and -1 is allowed.

If a file of 'name.hlp' exists, it is assumed to be a 'help file' which will be printed to the screen if the camera operator invokes the command with the wrong arguments.

-clear Clears all previous runcmd definitions.

EXAMPLES

```
runcmd lineup .\run\lineup.run 0 # Setup a 'lineup' script

runcmd woff .\run\w.run 1        # Setup 'woff' (windoff) command
                                 # that expects 1 argument.

runcmd woff - 0                  # Delete any previous 'woff' command
```

DESCRIPTION

RUNCMD(OPCS) lets you define your own commands as run scripts, with optional help text for users if they specify the wrong number of arguments. These commands will be recognized as if they were built in to the OPCS program.

Whenever the OPCS software reads a command from the operator, it checks to see if the [name] for any RUNCMD matches a command the operator entered. If so, OPCS will use [filename] as the name of a RUN script to execute in place of the command. Arguments can be specified by the operator if the RUNCMD and the associated script file are set up for it. If the user does not specify all the arguments, the text in the help file (name.hlp) will be displayed, if the file exists.

All techniques that can be used in a RUN(OPCS) script can be used in scripts defined with RUNCMD(OPCSDEFS). Additionally, the $ argument mechanism allows the passing of arguments to run scripts.

Commands defined with RUNCMD can be stacked on one line like any other OPCS commands, as long as the number of arguments is not -1 (i.e. not 'variable'). If variable args are used, then commands CANNOT be stacked.

RUNCMD(OPCSDEFS) defined commands will show up in the '?' help listing contained in parentheses, indicating they are commands not part of the software. DOSCMD(OPCSDEFS) definitions also show up this way.

OPTIONAL HELP FILES (K1.13a+)

When a runcmd is invoked without the proper number of arguments, an error message will be printed. If you want, you can also have some help text printed.

To make a 'help file' for the command, create a file in the same directory, and of the same name as the runcmd's filename, but with a .HLP extension. If the file exists, the files contents are displayed along with the error. Consider the following runcmd:

```
runcmd foo /opcs/mystuff/foo.run 3
```

In this case, the optional help file would be called:

```
/opcs/mystuff/foo.hlp
```

The file should just contain ascii text that is printed verbatim to the user's screen following the 'bad/missing arguments' error. This should probably just be a few short lines of text with a description of the command and its expected arguments.

LIMITS

Currently, no more than 30 different RUNCMD definitions can be setup. There is a 10 character limit on [name], an 80 character limit on [filename], and anywhere from 0 to 9 arguments are allowed per command.

ACTUAL EXAMPLES

.\SCRIPTS\LINEUP.RUN:

```
@go c 1000     # seat camera for lineups
# CAMERA IS NOW SEATED FOR SMPTE LINEUP
@pse           # pause while operator does lineup
@go c -1000    # unseat, without loosing frame position
```

Note the use of the '@' prefix. This allows the commands to execute without echoing to the screen. See RUN(OPCS) for more on the '@' prefix.

ARGUMENTS

The following example shows how to pass arguments from the command line to a script using the $ mechanism. $ followed by a digit 1-9 is replaced by arguments specified on the operator's command line, and $* is replaced by ALL arguments that were supplied. (Note: Use $$ to specify an actual '$' to prevent it being interpreted as an argument variable)

.\run\w.run might look like:

```
# RUNNING OFF $1 FRAMES WITH SHUTTER CLOSED
@! echo seekcap yes > foo.foo !  ldefs foo.foo
@seek $1
```

If 'runcmd w .\run\w.run 1' is defined in the OPCSDEFS.OPC file, when the operator types 'w 80', the script file will execute, and the argument 80 will replace all occurrences of $1 in the script file. The resulting script will automatically be interpreted as the following during execution:

```
# RUNNING OFF 80 FRAMES WITH SHUTTER CLOSED
@! echo seekcap yes > foo.foo !  ldefs foo.foo
@seek 80
```

You can specify up to 9 arguments if the script file and RUNCMD are setup for it. Here is an example that uses two arguments in a cross dissolve command. The first argument is the number of cross dissolves, the second argument is the number of frames in each cross dissolve:

---- OPCSDEFS.OPC:

```
runcmd xdx .\run\xdx.run 2     # X-dissolve command
```

---- .\RUN\XDX.RUN:

```
# DOING $1 CROSS DISSOLVE(S) $2 FRAMES EACH
do $1  dxo $2 cam $2 cam -$2 pro ($2+20) dxi $2 cam $2
```

When the operator executes 'xdx 70 8', the script is interpreted:

```
# DOING 70 CROSS DISSOLVE(S) 8 FRAMES EACH
do 70  dxo 8 cam  cam -8 pro 28 dxi 8 cam 8
```

Here's an example that uses variable arguments:

---- OPCSDEFS.OPC:

```
runcmd zoom .\run\zoom.run -1    # setup a zoom
```

---- .\RUN\ZOOM.RUN:

```
! ease foo.tmp $*
feed e foo.tmp
```

In this case, any number of arguments can be specified to 'zoom', and will be expanded into the script where the $* is specified.

GOTCHYAS

The RUNCMD can seem really nice at first, but it does have drawbacks.

You are basically 'customizing' a system when you add such definitions, which can be as bad as 'creating a new version' of the software. Symptoms will be:

- Operators used to a plain-jane OPCS system will be confused by commands they have never seen before.

- RUN scripts containing commands that are really defined by RUNCMD will cause errors on systems that don’t have the same definitions.

BUGS

OPCS does not do any argument type checking when arguments are passed through commands defined by RUNCMD. If the user forgets to specify an argument, such as with the 'w' command in the following example:

```
w pro 12       # user forgot #frames following 'w'
```

'pro' will be passed as an argument to 'w', and the executing script will fail with an error because 'pro' will eventually be used as a frame argument to the SEEK command, which will cause a somewhat confusing error:

```
seek: bad or missing argument
Stopped at line 2 of 1: .\run\w.run
```

HISTORY

The -clear option was added in K2.02.

SEE ALSO

RUN(OPCS)	- run a script file
DOSCMD(OPCSDEFS)	- define DOS commands to the OPCS software
CUSTOM(OPCS)	- how to make your own 'custom' opcs commands
LOAD(OPCS)	- 'load' is really a custom runcmd
LINEUP(OPCS)	- 'lineup' is really a custom runcmd
UNLOCK(OPCS)	- 'unlock' is really a custom runcmd

ORIGIN

Gregory Ercolano, Los Feliz California 04/19/91
$1 and $* notation used in the UNIX Bourne and C Shells.

```
SAMPSPERSEC(OPCSDEFS)   Optical Printer Control System    SAMPSPERSEC(OPCSDEFS)

    NAME
        sampspersec - Sets the motion control card's samples per second

    USAGE
        sampspersec <float>    # floating point number of
                               # velocity samples per second

    EXAMPLES
        sampspersec 107.0      # A800 card (REV A) - IRQ5 is 107Hz
        sampspersec 120.0      # A800 card (REV B) - IRQ5 is 120Hz
        sampspersec 120.0      # RTMC16 and RTMC48 - IRQ is 120Hz

    DESCRIPTION
        Sets the number of velocity samples per second the motion control
        card uses for its velocity values. This is the same as the IRQ
        rate in Hertz.

            > All Kuper cards use 120.0 (RTMC16, RTMC48, Kuper Industrial)

            > The OPCS "A800" board uses:
                    107.0 -- for REV A firmware
                    120.0 -- for REV B firmware

        "Samples" are the term used for the velocity sample rate
        in the stepper pulse generator cards.

        A velocity value is the number of step pulses sent per sample.
        So if the sample rate is 120 per second:

                Velocity Value          Motor Speed
                --------------          -----------------------
                      1                 120 steps per second
                      2                 240 steps per second
                      :
                      255               30,600 steps per second
                      :

        Put another way:

            > A velocity of '1' sends one step  during the sample time
            > A velocity of '2' sends two steps during the sample time

        So basically the sample rate times the velocity gives you the
        number of steps per second. Most microstepping motors are
        configured to run at 2000 pulses per revolution (PPR).

        To compute revs per second (RPS):

                (sampspersec * velocity) / PPR = RPS

        To compute revs per minute (RPM):

                (sampspersec * velocity) / PPR / 60 = RPM
```

HISTORY

For decades only the Kuper Controls cards were supported by OPCS, which are all 120 samples per second, and this value was configured in the old (and now obsolete) STARTUP.DEFS file.

In K2.00, 'sampspersec' from STARTUP.DEFS was moved here to OPCSDEFS.OPC. This value is used to support the new A800 board, which can have different sample rates depending on the firmware.

ORIGIN

Version K2.00 Gregory Ercolano, Alhambra California 06/01/20

```
SEEKCAP(OPCSDEFS)       Optical Printer Control System      SEEKCAP(OPCSDEFS)

    NAME
        seekcap - configure the fader to cap during SEEK commands

    USAGE
        seekcap [on/off]        # ON enables automatic capping

    EXAMPLES
        seekcap on    # cap fader when seek is used with camera
        seekcap off   # no auto cap

    DESCRIPTION
        This command enables automatic capping of the fader whenever the
        camera is moved with the SEEK command.

        Default is 'on'.

    SEE ALSO
        SEEK(OPCS) - seek to positions quickly on camera/projector(s)

    ORIGIN
        Gregory Ercolano, Los Feliz California 7/28/91
```

```
SETBIT(OPCSDEFS)        Optical Printer Control System    SETBIT(OPCSDEFS)

    NAME
        setbit - set bit(s) on a port

    USAGE
        setbit [port] [mask] [softlatch]    # (values hex!)

    EXAMPLES
        setbit 0378 04 0        # lpt1 port bit #2 (1=#0, 2=#1, 4=#2)
        setbit 0306 01 1        # kuper card logic connector,
                                # bit #1 (note softlatch=1)

    DESCRIPTION
        This command enables bits on port, based on the mask.
        Basically, the mask is OR'ed with the port's current value.

        [port] is the port number in hex.

        [mask] is a hex byte value which is OR'ed with the current
        value at the port. (In the case of softlatching, the software latched
        value is OR'ed with the mask, to create the new value that it output
        to the port)

    CAVEATS
        o With [softlatch] set to 1, only ports 0x0000 - 0x07ff are allowed.
          Any ports above 0x07ff with [softlatch] enabled causes an error.

        o External programs changing port bits defined to OPCS with [softlatch]
          (e.g. the Kuper logic I/O port) should be aware that OPCS is
          maintaining its own internal latch for that port, and that latch
          won't know about hardware changes made by external programs.

        o Due to these issues, it's best to avoid using hardware that has to
          be latched. It's usually bad hardware practice to make WRITE ONLY
          ports, since different programs cannot co-communicate with them,
          unless some common data area or driver is arranged.

    SEE ALSO
        DEENERGIZE(OPCSDEFS)  - define port/bit to deenergize motors
        ALLSTOP(OPDSDEFS)     - define port/bit to detect the allstop key
        BUCKLE(OPCSDEFS)      - define port/bit to detect film buckles
        VIEWER(OPCSDEFS)      - define port/bit to detect viewer open
        TRIPSWITCH(OPCSDEFS)  - define port/bit to detect trip switches
        SETBIT(OPCSDEFS)      - set bit(s) on a port
        CLRBIT(OPCSDEFS)      - clear bit(s) on a port
        XORBIT(OPCSDEFS)      - invert bit(s) on a port

    ORIGIN
        Version K1.12d+ Gregory Ercolano, Venice California 03/04/98
```

SLOP(OPCS) Optical Printer Control System SLOP(OPCS)

NAME
slop - configure 'slop correction' for sloppy hardware (e.g. faders)

SYNOPSIS
slop [chan] [steps]

EXAMPLES
slop d 300 # indicates the fader has 300 steps of 'slop'

DESCRIPTION
SLOP tells the software to take up slop for a motor whenever it is told to run in a prescribed direction.

The sign of the [steps] arguments tells the software which direction it should prefer to take up slop in. A positive number takes up slop when the motor moves in a positive direction, a negative number takes up slop in the negative direction.

To determine how much slop a motor has, disable any slop commands for the motor. Use JOG(OPCS) to move the motor in one direction. Now change directions, making note of how many steps you can tell the computer to run in the new direction before the equipment starts to actually move. Use this number of steps in the SLOP(OPCSDEFS) command for that motor, and note how the software tries to take up the slop.

If the motor is being moved by a command in the direction the software wants to take up slop, the software will move the motor that many pulses BEYOND the position requested, and then back that many pulses to take up slop. This technique ensures the equipment is always resting on the same edge of the sloppy equipment, which can allow accurate positioning of even the sloppiest mechanics.

NOTES
For those of you who think it is a waste to have to take up slop EACH TIME the motor turns in the predefined direction, and that 'it should only take up slop once..when it changes direction', think again, pal. You are assuming the slop distance is a fixed entity, which it rarely is.

In order to arrive at positions properly, the position must be found by always leaving off having moved to the position FROM THE SAME DIRECTION, so that the equipment is always left off resting on the same side of the equipment slop. This necessitates always doing the double-move slop take up whenever the motor moves in one of the two directions.

BUGS
none.

SEE ALSO
INTERP(OPCSDEFS)
GO(OPCS)

ORIGIN
Gregory Ercolano, Los Feliz California 12/15/89

SPD(OPCSDEFS) Optical Printer Control System SPD(OPCSDEFS)

NAME

spd - configure the default and fastwind/slewing speeds for a motor

SYNOPSIS

spd [chan] [normal] [fast] [scale] [offset]

NOTE: any of the arguments [normal] [fast] [scale] [offset] can be a dash (-), indicating that argument won't be modified.

EXAMPLES

```
spd c .25 .1 1.0 0.0  # sets default speed for the camera motor:
                      #  .25 is normal running speed,
                      #  .10 is the slew speed (used by SEEK)
                      #  1.0 and 0.0 indicate speeds are specified
                      #      as ROTATIONAL speeds.

spd c .25 .1 3.0 0.0  # Same as above, but shows correct [scale]
                      #  value so EXPOSURES can be specified
                      #  for a 120 deg. shutter (3.0 = 360/120).

spd c .25 .1 2.1176   # Same as above, but shows correct [scale]
                      #  value so EXPOSURES can be specified
                      #  for a 170 deg. shutter (2.1176 = 360/170)
```

DESCRIPTION

Normally, one of these commands for EACH motor should appear in the OPCSDEFS.OPC file. This command sets the initial running speeds for a motor, as well as how speeds are specified (i.e., rotational or exposure speeds. See below.)

This command also allows SCALEs and OFFSETs to be applied to speed values automatically to let you specify EXPOSURE speeds instead of rotational speeds...

ROTATIONAL SPEED

An example of a rotational speed could be .25, which would mean: a full rotation of the camera shaft will occur in .25 of a second, or 1/4 a second. This is how motor speeds are normally handled by the software.

EXPOSURE SPEED

An example of an exposure speed of .5 would mean the film exposes to light for 1/2 second. Since most shutters are 170 degrees (i.e. exposing light for 170 degrees out of the total 360 degrees of rotation), the [scale] value can be used to compensate. You will want to decrease the camera's rotational speed. To do this, multiply 360/170 times the current speed to compensate for the fact that the shutter is only open for a fraction of a rotation.

This is where the [scale] argument comes in. By setting [scale] to 2.1176 (360/170), you can then specify speeds as 'exposure speeds', and the system will compensate automatically. If you have a 120 degree shutter, use 3.0 (360/120) for the [scale] value.

You need only do this for the camera motor..the projectors do not have to be set up this way, since the projectors always slave to the camera's speed whenever a tandem run is executed.

EQUATION

The following equation shows how scales and offsets are first applied to motor speeds:

actual motor speed = (norm_speed * scale) + offset * spdinterp

A value of 1.0 for [scale], and 0.0 for [offset] makes NO CHANGE in the norm_speed (normal running speed), and thus will reflect shaft rotation speed.

'spdinterp' will affect the equation only if a SPDINTERP(OPCSDEFS) command is configured, in which case the speed will be modified according to the current position of the SPDINTERP's master channel.

SEE ALSO

```
MRP(OPCSDEFS)     - set 'maximum ramping pulses' for shutter runs
RAMP(OPCSDEFS)    - set maximum accelerations and velocities
SPD(OPCS)         - set the camera's exposure speed
SPD(OPCSDEFS)     - set a motor's running speeds
RAMPCURVE(OPCSDEFS) - set ramping curves for shutter runs
SPDINTERP(OPCSDEFS) - set auto-interpolation for exposure speeds
```

BUGS

Speeds of 0.0 will cause the software to blow out unpleasantly. Avoid setting a motor's speed to zero, or doing any operation that would result in an actual speed of zero.

ORIGIN

Gregory Ercolano, Los Feliz California 11/29/89

SPDINTERP(OPCSDEFS) Optical Printer Control System SPDINTERP(OPCSDEFS)

NAME

spdinterp - configure exposure speed interpolations

SYNOPSIS

spdinterp [slavechan] [masterchan] [lowpos] [highpos] [total] [samples]

EXAMPLES

spdinterp c e -1000 1000 2 0.5 1.5

```
          - - ----------  -  -------
          | |      |      |     |
          | |      |      |     The exposure compensation values.
          | |      |      |
          | |      |      # of exposure compensation values.
          | |      |
          | |      The extreme positions for the 'e' (zoom) channel.
          | |
          | 'e' (zoom) is the master channel.
          |
          'c' (camera) exposure speed slaved to [masterchan]
```

DESCRIPTION

SPDINTERP allows a channel's normal running speed to be affected by the position of some other channel. Slewing speeds are NOT affected.

[slavechan] is the channel whose exposure speed will be affected. This is usually always the 'c' (camera) channel.

[masterchan] is channel whose position will dictate the exposure speed of the [slavechan]. Normally, this is the zoom channel, so that moving the zoom will auto-compensate the exposure.

[lowpos] [highpos] are the low and high positions the [masterchan]; the extreme positions which define the range over which the interpolation will take place.

[total] The total number of exposure compensation sample values. NOTE: If [total] is '0', this will cancel any previous SPDINTERP(OPCSDEFS) specifications for the [slavechan] channel.

[samples] are the compensation values, which will be multiplied to [chan]'s current exposure speed, to create the actual, 'compensated' exposure speed.

A value of 2.0 will effectively double the exposure speed, 0.5 will cut the exposure speed by half, 1.0 will leave the exposure unmodified, etc.

Each sample position should be separated by white space (tabs, spaces, CRLFs) and there should be as many samples as specified by [total].

OVERVIEW

When an SPDINTERP is set up on a slave channel, it is like defining a look up table through which positions on the master channel affects the exposure speed of the slave channel.

Requests that fall between values in the lookup table are computed as a linear interpolation between the two neighboring lookup values.

The resulting interpolation results in a 'compensator value' that is simply multiplied to the current camera speed, to get the compensated 'actual speed'.

HOW SPDINTERP WORKS

```
================================================================
=   spdinterp c e -1000 1000 3 .5 1.0 2.0                      =
=                 |      |  |  |     |                         =
=                 |      |  |  |     Last sample position      =
=                 |      |  |  First sample position           =
=                 |      |  Total samples                      =
=                 |      High                                  =
=                 Low                                          =
================================================================
```

Figure A.

```
=========================================================================
=  Zoom        Clip                          Interped    Compensated   =
=  Posns      Window         Samples         Compensator Actual Speed  =
=                                            Values      (SPD=0.5)     =
=              |                                                        =
=  -2000 ---->  |                          ----- 0.50 ----> 0.25        =
=  -1500 ---->  | LOW = -1000             /------ 0.50 ----> 0.25       =
=  -1000 ----------------------> 0.5 -------- 0.50 ----> 0.25        =
=   -500 ---------------------->     -------- 0.75 ----> 0.38        =
=      0 ----------------------> 1.0 -------- 1.00 ----> 0.50        =
=    500 ---------------------->     -------- 1.50 ----> 0.75        =
=   1000 ----------------------> 2.0 -------- 2.00 ----> 1.00        =
=   1500 ---->  | HIGH = 1000        \------ 2.00 ----> 1.00        =
=   2000 ---->  |                          ----- 2.00 ----> 1.00        =
=              |                                                        =
=========================================================================
```

Figure B.

Whenever the zoom moves to a new position, it is clipped through the 'clipping window' between the [low] and [high]. This creates a lookup into the sample list using a linear interpolation. Results in an 'Interped Compensator Value', which is then multiplied by the current speed of the camera (SPD=0.5), resulting in 'Compensated Actual Speed'.

Note when zoom is 0 (exactly between the LO and HIGH values) the actual speed is the same as the camera speed; the compensator value becomes 1.0, which multiplied against the camera speed yields no change. This is because the 'zero' position on the zoom is usually 1:1, which is where exposure speeds should match the actual speed.

For this reason, it is best if the extreme zoom positions are totally opposite (2:1 vs 1:2, or 10:1 vs 1:10, etc), and the 'middle value' in the samples is 1.0. This way if zoom is 1:1, exposure is unchanged.

SEE ALSO

FEED(OPCS) - feed new positions to motors every camera frame
INTERP(OPCSDEFS) - set up interpolation points

ORIGIN

Version K1.12e+ Gregory Ercolano, Venice California 04/12/98

TENSION(OPCSDEFS) Optical Printer Control System TENSION(OPCSDEFS)

NAME

tension - configure tension motor port for each channel

USAGE

```
tension [chan] [port] [mask] [invert] [softlatch]    # (values hex!)
```

EXAMPLES

```
tension a 03bc 01 00 0        # lpt1 output port bit #1
tension a 0306 01 00 1        # RTMC16 logic connector bit #1
                              # (note [softlatch] set to 1)
```

DESCRIPTION

Defines the port/bit combo that controls the film tension motors for when the specified [chan] changes direction. Intended only for cam/pro/pro2 (channels a/b/c) to maintain proper tension on the film.

[chan] is the channel name of the motor to be affected.

[port] is the hard port. 0000 disables tension motor control.

[mask] is the bit on the port that manages the tension motor.

[invert] if '0', running forward clears the [port] bit in [mask], and running reverse sets it. If '1', the opposite occurs.

CAVEATS

- With [softlatch] set to 1, only ports 0x0000 - 0x07ff are allowed. Any ports above 0x07ff with [softlatch] enabled causes an error.

- External programs changing port bits defined to OPCS with [softlatch] (e.g. the kuper logic I/O port) should be aware that OPCS is maintaining its own internal latch for that port, and that latch won't know about hardware changes made by external programs.

- Due to these issues, it's best to avoid using hardware that has to be latched. It's usually bad hardware practice to make WRITE ONLY ports, since different programs cannot co-communicate with them, unless some common data area or driver is arranged.

SEE ALSO

```
DEENERGIZE(OPCSDEFS)  - define port/bit to deenergize motors
ALLSTOP(OPDSDEFS)     - define port/bit to detect the allstop key
BUCKLE(OPCSDEFS)      - define port/bit to detect film buckles
VIEWER(OPCSDEFS)      - define port/bit to detect viewer open
TRIPSWITCH(OPCSDEFS)  - define port/bit to detect trip switches
SETBIT(OPCSDEFS)      - set bit(s) on a port
CLRBIT(OPCSDEFS)      - clear bit(s) on a port
XORBIT(OPCSDEFS)      - invert bit(s) on a port
```

ORIGIN

Gregory Ercolano, Los Feliz California 09/11/90

TRIPSWITCH(OPCSDEFS) Optical Printer Control System TRIPSWITCH(OPCSDEFS)

NAME

tripswitch - configure trip switches for axes

USAGE

```
tripswitch [port] [mask] [test] [0]   # (all values in HEX!)
```

EXAMPLES

```
tripswitch 0000 00 00 0      # disable all trip switch detection
tripswitch 03bd 40 00 0      # LPT1 pin 10, HI bit detects condition
tripswitch 03bd 40 40 0      # LPT1 pin 10, LO bit detects condition
```

DESCRIPTION

Trip switches are set up to stop motion control moves (FEED, GO, etc) if an axis is about to go off its track. When a trip occurs, the software will stop the motors and indicate a trip error:

**** TRIP SWITCH ERROR - AXIS WENT TOO FAR ****
RETURN to continue, or SPACEBAR to ABORT:

Sensors should be wired so that a light by the trip switch indicates which trip switch was activated, so the problem can be corrected.

A trip condition will occur only when any switch changes from its normal state to its trip state. A trip condition will NOT occur when the switch changes back to its normal state. This allows the operator to back the motor off after a trip occurred, without causing recurring trip errors.

[port] is the port number in the range 0000-03ff. If [port] is 0000, no trip switch checking is done.

[mask] is applied to the value received from the port whenever the software is checking for a trip switch condition. This is applied before comparing to [test].

[test] is compared to the value read from the port after [mask] is applied. If the result is the same as [test], a trip condition will occur.

[0] is always zero.

WIRING CONSIDERATIONS

Normally you would use an optically isolated interface card for the buckle and viewer switches, e.g. PIO-100(DOCS).

If directly connecting switches to the parallel port, use a separate dedicated 5 VDC power supply wired through the switches such that when the sensing switch is tripped, +5V is supplied to the computer.

Such a supply can be a store-bought 5 VDC regulated switching transformer, which gives out *exactly* 5VDC. (Avoid older linear transformers, as they are often unregulated)

As with any signal going to the sensing input on a computer, the signal should never be open. The signal must pull either 5 volts or ground for a TRUE or FALSE condition. Open inputs can act like radio antennas that will oscillate randomly.

If you have noise problems, shielded wire can help mitigate noise. Shield ONLY at the power supply end. Ground the shield to chassis ground if possible. Keep wire lengths as short as possible.

If noise problems persist, it may be that the wire is simply too long for such a low voltage signal. The PIO-100 uses 12VDC to allow for longer cable lengths, and an optoisolator to convert to +5V for the parallel port input.

You can find the base port value for the parallel ports from the operating system using the DOS 'debug' utility:

```
C>debug                                 # run 'debug'
-d40:8 f                                # enter this (not the '-')
0040:0008  BC 03 78 03 00 00 00 00      # debug spits this out
-q         ----- -----                  # type 'q' to quit debug
             |     |
             |     LPT #2's  port base address
             |
             LPT #1's  port base address
```

Your machine may show different values. In the case above, 03BC is the base port value for LPT1..note the bytes are in reverse order in typical LSB/MSB fashion.

See the PARALLEL() man page which shows the pin out and port addresses of the IBM PC's parallel ports.

The following shows an example of how to wire the trip switches.

AT THE PARALLEL CONNECTOR

Using a shielded 2 conductor cable, wire the dark conductor to the computer's parallel port input bit. Then run a 400 ohm resistor from the input pin to one of the GND pins (a pull down configuration).

Wire the light conductor to the +5 volt supply. Ground the shield AND the +5 volt supply's ground to all of the parallel port's ground pins (18-25).

AT EACH SPDT TRIP SWITCH

Throughout the wire, you can drop in single pole dual terminal micro switches. Wire the switch's Normally Open (NO) terminal to the dark conductor that goes to the computer. Wire the switch's Common (C) to the light conductor (+5). Wire an LED such that the negative input is to the Normally Closed (NC) terminal, and the positive input is connected to C terminal along with the light conductor (+5). Now tie a 400 ohm resistor from the NC terminal to the shield (ground).

Once wired, the input to the computer will be normally grounded (through the 400 ohm resistor at the connector). All the LEDs will be dark, because they are seeing 5 volts across their terminals, and 5 volts will leak through the associated 400 ohm resistor.

If a switch is tripped, 5 volts is brought to the computer's input. The switch's LED will light because now ground is supplied to its negative terminal through the 400 ohm resistor, since NC is no longer shorted to +5 by the switch.

Set up the OPCSDEFS.OPC file to contain a tripswitch command with the appropriate port and bit information. Then enter the software and run one of the positional axes.

If a trip condition occurs only when the switch is RELEASED (instead of pressed), you don’t have to change the wiring of the switches..just change the [test] value. EXAMPLE:

```
    tripswitch 03bd 40 40 0     # If this doesn't work...
    tripswitch 03bd 40 00 0     # Try this, or vice-versa.
                       --
```

If you find that it does not matter whether a switch is pressed or not, then the port and/or mask bits may be wrong. You can use the 'PARALLEL.EXE' utility to monitor a parallel port's pins, so you can see which bit is actually changing when you press a trip switch. When executing 'parallel' from DOS, you can specify which port you want to monitor:

```
        parallel 1    # This will monitor LPT1
        parallel 2    # This will monitor LPT2
```

SEE ALSO

```
    DEENERGIZE(OPCSDEFS)  - define port/bit to deenergize motors
    ALLSTOP(OPDSDEFS)     - define port/bit to detect the allstop key
    BUCKLE(OPCSDEFS)      - define port/bit to detect film buckles
    VIEWER(OPCSDEFS)      - define port/bit to detect viewer open
    TRIPSWITCH(OPCSDEFS)  - define port/bit to detect trip switches
    SETBIT(OPCSDEFS)      - set bit(s) on a port
    CLRBIT(OPCSDEFS)      - clear bit(s) on a port
    XORBIT(OPCSDEFS)      - invert bit(s) on a port
    PARALLEL(BIOS)  - parallel port pinout with port/bit masks
```

ORIGIN

Gregory Ercolano, Los Feliz California 10/11/90

VIEWER(OPCSDEFS) Optical Printer Control System VIEWER(OPCSDEFS)

NAME

viewer - configure the viewer input port and bit mask

USAGE

viewer [port] [mask] [test] [0] # (all values in hex!)

EXAMPLES

viewer 0000 00 00 # No viewer detection
viewer 03bd 40 40 # LPT1 pin 10, HI bit detects condition
viewer 03bd 40 00 # LPT1 pin 10, LO bit detects condition

DESCRIPTION

If your system has a 'viewer open' switch, it can be wired to one of the IBM parallel ports to allow the software to sense its state, to prevent exposing film with the viewer open.

[port] is the port number in the range 0000-03ff.
If [port] is 0000, no viewer checking is done.

[mask] is applied to the value received from the port whenever the software is checking for a viewer open condition. This is applied before comparing to [test].

[test] is compared to the value read from the port after [mask] is applied. If the result is the same as [test], a viewer open condition exists.

WIRING CONSIDERATIONS

Normally you would use an optically isolated interface card for the buckle and viewer switches, e.g. PIO-100(DOCS).

If directly connecting switches to the parallel port, use a separate dedicated 5 VDC power supply wired through the switches such that when the sensing switch is tripped, +5V is supplied to the computer.

Such a supply can be a store-bought 5 VDC regulated switching transformer, which gives out *exactly* 5VDC. (Avoid older linear transformers, as they are often unregulated)

As with any signal going to the sensing input on a computer, the signal should never be open. The signal must pull either 5 volts or ground for a TRUE or FALSE condition. Open inputs can act like radio antennas that will oscillate randomly.

If you have noise problems, shielded wire can help mitigate noise. Shield ONLY at the power supply end. Ground the shield to chassis ground if possible. Keep wire lengths as short as possible.

If noise problems persist, it may be that the wire is simply too long for such a low voltage signal. The PIO-100 uses 12VDC to allow for longer cable lengths, and an optoisolator to convert to +5V for the parallel port input.

You can find the base port value for the parallel ports from the operating system using the DOS 'debug' utility:

```
C>debug                                 # run 'debug'
-d40:8 f                                # enter this (not the '-')
0040:0008  BC 03 78 03 00 00 00 00      # debug spits this out
-q         ----- -----                  # type 'q' to quit debug
             |     |
             |     LPT #2's  port base address
             |
             LPT #1's  port base address
```

Your machine may show different values. In the case above, 03BC is the base port value for LPT1..note the bytes are in reverse order in typical LSB/MSB fashion.

See the PARALLEL() man page which shows the pin out and port addresses of the IBM PC's parallel ports.

BUGS

None.

SEE ALSO

DEENERGIZE(OPCSDEFS)	- define port/bit to deenergize motors
ALLSTOP(OPDSDEFS)	- define port/bit to detect the allstop key
BUCKLE(OPCSDEFS)	- define port/bit to detect film buckles
VIEWER(OPCSDEFS)	- define port/bit to detect viewer open
TRIPSWITCH(OPCSDEFS)	- define port/bit to detect trip switches
SETBIT(OPCSDEFS)	- set bit(s) on a port
CLRBIT(OPCSDEFS)	- clear bit(s) on a port
XORBIT(OPCSDEFS)	- invert bit(s) on a port
PARALLEL(BIOS)	- parallel port pinout with port/bit masks
PIO-100(DOCS)	- OPCS Parallel I/O interface board, e.g. http://seriss.com/opcs/pio-100/

ORIGIN

Version K1.12e+ Gregory Ercolano, Venice California 04/10/98

```
XORBIT(OPCSDEFS)        Optical Printer Control System    XORBIT(OPCSDEFS)

    NAME
        xorbit – invert bit(s) on a port using exclusive-or (XOR)

    USAGE
        xorbit [port] [mask] [softlatch]    # (values hex!)

    EXAMPLES
        xorbit 0378 04 0     # flip lpt1 port bit #2 (1=#0, 2=#1, 4=#2)
        xorbit 0306 01 1     # kuper card logic connector,
                             # flip bit #1 (note softlatch=1)

    DESCRIPTION
        This command flips bits on a port, based on the mask.
        All bits specified by the mask are inverted. (set will be clear,
        clear will be set, etc)

        [port] is the port number in hex.

        [mask] is a hex byte value indicating the bits to be flipped
        on that port.

    CAVEATS
        o With [softlatch] set to 1, only ports 0x0000 - 0x07ff are allowed.
          Any ports above 0x07ff with [softlatch] enabled causes an error.

        o External programs changing port bits defined to OPCS with [softlatch]
          (e.g. the kuper logic I/O port) should be aware that OPCS is
          maintaining its own internal latch for that port, and that latch
          won't know about hardware changes made by external programs.

        o Due to these issues, it's best to avoid using hardware that has to
          be latched. It's usually bad hardware practice to make WRITE ONLY
          ports, since different programs cannot co-communicate with them,
          unless some common data area or driver is arranged.

    SEE ALSO
        DEENERGIZE(OPCSDEFS)  - define port/bit to deenergize motors
        ALLSTOP(OPDSDEFS)     - define port/bit to detect the allstop key
        BUCKLE(OPCSDEFS)      - define port/bit to detect film buckles
        VIEWER(OPCSDEFS)      - define port/bit to detect viewer open
        TRIPSWITCH(OPCSDEFS)  - define port/bit to detect trip switches
        SETBIT(OPCSDEFS)      - set bit(s) on a port
        CLRBIT(OPCSDEFS)      - clear bit(s) on a port
        XORBIT(OPCSDEFS)      - invert bit(s) on a port

    ORIGIN
        Version K1.12d+ Gregory Ercolano, Venice California 03/04/98
```

MISCELLANEOUS DOCS

INTRO(DOCS) Optical Printer Control System INTRO(DOCS)

This section has general documentation ("DOCS") on OPCS related tutorials, hardware/software setup, tools, and other miscellany.

To get a list of all the OPCS documentation sections:

```
man -k OPCS:     -- All the OPCS commands
man -k OPCSDEFS: -- All the OPCSDEFS file setup commands
man -K DOCS:     -- Miscellaneous OPCS documentation (below)
```

When you use 'man -k DOCS:', you should see a list something like the below.

To read the full documentation for any item, use '**man <itemname>**', where **<itemname>** is the word in the left column below, e.g. '**man math**', '**man opcs**', '**man gecko**', etc.

Tutorials/Instructions

```
opcsetup  - Setting up OPCS software for the first time
opcshard  - Setting up OPCS hardware
quickref  - Camera Operator's quick reference for OPCS commands
```

Software/General

```
ease      - Ease in / Ease out move generator
error     - OPCS error messages
home      - OPCS home program
math      - Specifying math operations in OPCS commands
mov       - Create and edit moves (zoom, pan, etc)
opcs      - Running OPCS from the command line
opcsdefs  - The OPCS setup definitions file format/description
parallel  - Parallel port software monitor tool + pinout
versions  - OPCS Version information
syntax    - Online calculator and OPCS math expression syntax
```

Hardware

```
8255      - Controlling 8255 based I/O cards
a800      - Docs for the A800 step pulse generator board (OPCS)
centent   - Centent motor drive wiring
connector - Misc connector and card wiring used on older systems
CIODIO24  - CIO-DIO24 Digital I/O Board (3rd Party)
gecko     - Gecko motor drive wiring (3rd Party)
kuper     - Docs for the Kuper card, general wiring
pio-100   - Docs for the PIO-100 parallel I/O interface board
rtmc16    - RTMC16 step pulse generator board (Kuper Controls)
rtmc48    - RTMC48 step pulse generator board (Kuper Controls)
serial    - Serial port
slosyn    - SLOSYN motor hookups (centent/anaheim)
sd-800    - Stepper Distribution 8-channel board
            This breaks out the DB-37 connector into separate
            RJ-45 patch cables to each stepper drive
wiring    - Miscellaneous wiring diagrams
```

ORIGIN

Gregory Ercolano, Los Feliz California 08/17/20

NAME

8255 - how to control 8255 based digital I/O cards

8255 PORTS

The 8255 family of chips (82C55, etc) are usually one chip solutions to getting 24 bits of programmable digital I/O.

Typically, the chip is configured at a base I/O address, such as 0310.

There are four ports (base+0, base+1, base+2 and base+3) that are used to control the chip:

```
base+0    - PORT #A
base+1    - PORT #B
base+2    - PORT #C
base+3    - CONTROL REGISTER
```

8255 PROGRAMMING

The "control register" controls the I/O direction of the 3 ports, and breaks out as follows:

```
Bit   Description
---   -------------------------------------------------
 0    Port C (low 4 bits): 1=input, 0=output
 1    Port B (all 8 bits): 1=input, 0=output
 2    Mode selection: 0=MODE#0, 1=MODE#1
 3    Port C (hi 4 bits): 1=input, 0=output
 4    Port A (all 8 bits): 1=input, 0=output
   _
 5  |_ 00 = MODE#0 (basic I/O)
 6  _|  01 = MODE#1 (strobed I/O)
        1x = MODE#2 (bidirectional bus)

 7    Mode set flag (1=active, 0=normal)
```

No initialization is required to achieve 24 bits of input, which is the default; during reset, all ports are programmed to be inputs.

The control register must be programmed before doing any I/O with the three ports A,B and C. Example values for the Control Register:

```
0x80    - A,B,C outputs
0x9b    - A,B,C inputs
0x92    - A+B input, C=output
```

8255 PROGRAMMING EXAMPLES

Here's a C programming example that shows how to setup the 8255 such that A+B are inputs, and C is outputs:

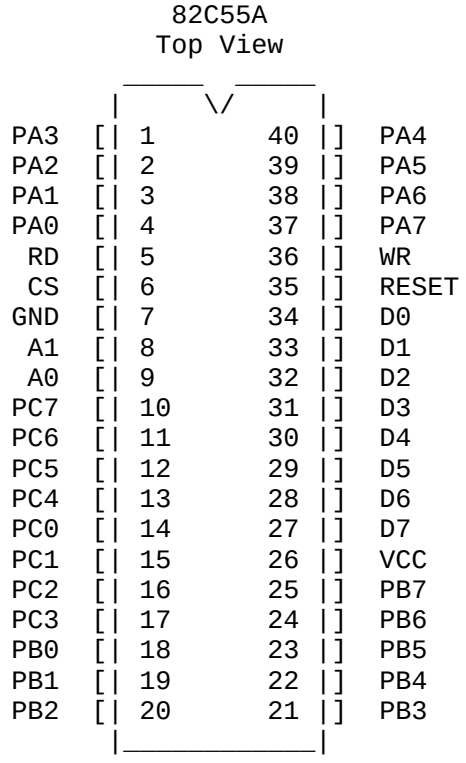
```
/* INITIALIZATION */
base = 0x310;
out(base+3), 0x92;      /* A+B=in, C=out */
/* READ/WRITE */
if ( inp(base+0) & 0x01 )         /* read A */
    printf("Bit #1 set on port A\n");
else
    printf("Bit #1 clear on port A\n");
/* SET PORT C, BIT #1 */
out(base+2, inp(base+2) | 0x01 ); /* write C */
```

This example shows similar 8255 programming example in the OPCS system's HOMEDEFS.HOM file, used by the HOME program:

```
# homedefs.hom
start init_8255
{
    # INITIALIZATION
    outport 0313 92        # A+B=in, C=out

    # READ/WRITE
    portset? 0310 01       # test if bit 1 set on Port A
    {
        print "Bit #1 set on Port A"
    }
    portclr? 0310 01       # test if bit 1 clear on Port A
    {
        print "Bit #1 clear on Port A"
    }

    # SET PORT C, BIT #1
    setbit 0312 01
}
end init_8255
```

8255 CHIP PINOUT

The 82C55A is most commonly found in a 40 pin DIP.



8255 PORT MONITOR PROGRAM

The OPCS software comes with 8255.exe which can monitor the real time status of the 8255's I/O ports.

Run '8255.exe' with the hex base address as the first argument:

```
C:\OPCS\WORK> 8255.EXE 0310
              -------- ----
                  |     |
                  |     Base address argument
                  |
                  Runs the 8255 program, usually in
                  the \OPCS\BIN directory.
```

This tool can be downloaded from https://seriss.com/opcs/ftp/ and the source code on githib at https://github.com/erco77/8255-dos/

CONTROL REGISTER: BASE+3

The control register is a single 8 bit port whose byte defines the I/O direction of all three 8 bit I/O ports A,B,C, and the mode for that I/O, modes 0,1 and 2 are supported by the 8255 chips.

The control register for each 8255 should be programmed on boot or OPCS startup to define whether ports A, B and C are inputs or outputs.

Initializing the control register can be done using the "home" program, creating a special entry in the HOMEDEFS.HOM file to initialize the port (see example below), and then invoking "**home 8255_init**" during the AUTOEXEC.BAT or via the OPCSDEFS.OPC setup file that's run when OPCS starts.

Example 8255 init using HOMEDEFS.HOM, where the base address for the 8255 is 0200:

1. Create a new entry called 'init_8255' in the HOMEDEFS.HOM file:

```
# Initialize the 8255 CIO/DIO board
start init_8255
{
    outport 0203 9b    # program the 8255 Control Register
}
end init_8255
```

   Replace 0203 with the base address+3 of your 8255 board, and replace '9b' with the Control Register value appropriate for your setup, as it defines which ports are input vs. output.

   See the table below "Control Register - "Mode 0" Operation" for a list of all the possible I/O configuration control register values.

2. Create an entry in either the AUTOEXEC.BAT file, or the batch file you use to start OPCS, using:

```
home init_8255
```

   ..which programs the 8255's control register based on the above HOMEDEFS.HOM file entry.

Note that individual bits can NOT BE programmed separately;

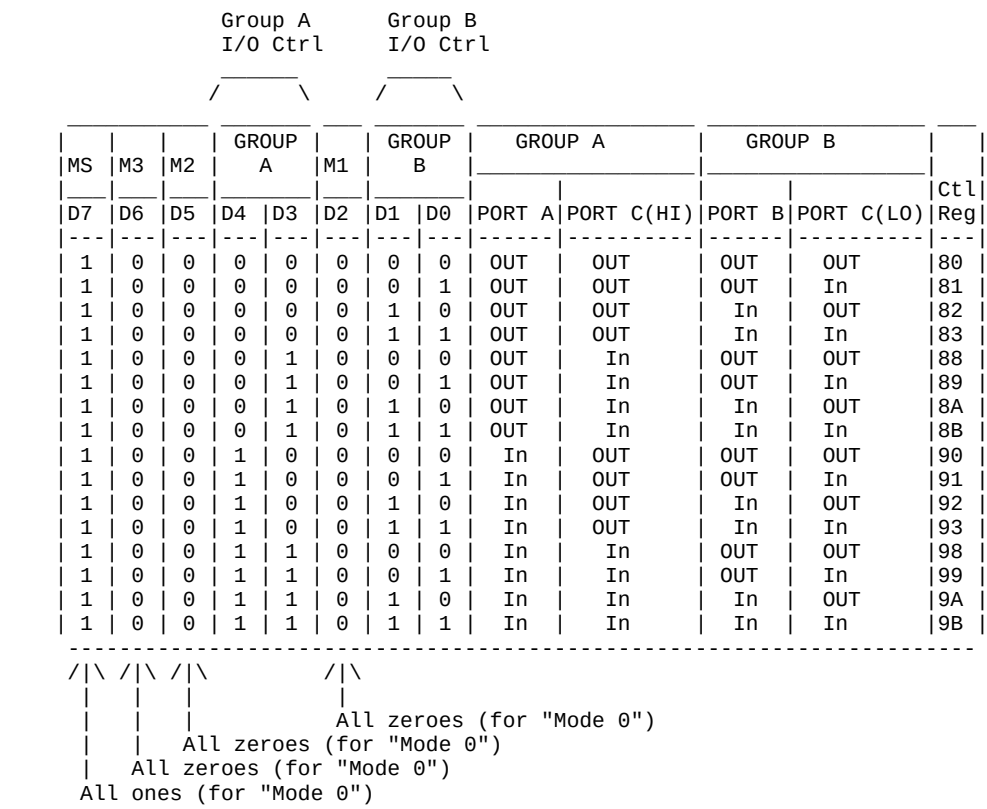
> Port A can be programmed to either be "all in" or "all out"
> Same for Port B
> Port C, the LOWER and UPPER 4 bits can be controlled separately for Input or Output.

I strongly suggest referring to the 8255 and/or 82C55 data sheets for authoritative information about how to program the 8255 for I/O. Secondarily the manual for the CIO-DIO48 ISA card.

What follows below is my reduction of those docs. (erco@seriss.com)

Control Register - "Mode 0" Operation

This document ONLY covers "Mode 0" of the 8255, which is simple "real time I/O" for all ports. Mode 1 (Strobed I/O) and Mode 2 (Bi-Directional Bus) are not covered here. For more info on those features, refer to the 8255 data sheet.



MS = Mode Set bit (hi bit of control register), must be 1 for "Mode 0"
M3,M2,M1 = must all be 0 for "Mode 0".

Here are all the possible 8 bit Control Register values (in HEX) to program Ports A,B,C in "Mode 0" for all possible Input/Output combinations:

Control Register Value (hex)	PORT A	(HI 4) PORT C	PORT B	(LO 4) PORT C	
80	Out	Out	Out	Out	<- Port A,B,C=Out
81	Out	Out	Out	In	
82	Out	Out	In	Out	
83	Out	Out	In	In	
88	Out	In	Out	Out	
89	Out	In	Out	In	
8A	Out	In	In	Out	
8B	Out	In	In	In	
90	In	Out	Out	Out	
91	In	Out	Out	In	
92	In	Out	In	Out	<- Port A,B=In, C=Out
93	In	Out	In	In	
98	In	In	Out	Out	
99	In	In	Out	In	
9A	In	In	In	Out	
9B	In	In	In	In	<- Port A,B,C=In

(The above table is from the CIO-DIO48 ISA board manual for the 82C55)

Put simply, the bit mask for the control register I/O control for Mode 0 for ports A,B,C is:

```
Data Bit  Mask  Description
--------  ----  -----------
   D0     01h   Port C(LO)
   D1     02h   Port B
   D3     08h   Port C(HI)
   D4     10h   Port A

NOTE: "D2" bit is skipped because it's the M1 mode bit
```

So if the base address is 0310, one can program ports A,B and C of the first 24 bits of I/O to all be inputs using the C code:

```
outp(0x0313, 0x9b);        // Program ports A,B,C to be inputs
```

And to read the input bits for ports A,B,C:

```
int a_bits = inp(0x0310);   // Read PORT A bits
int b_bits = inp(0x0311);   // Read PORT B bits
int c_bits = inp(0x0312);   // Read PORT C bits
```

The same can be done for the second 24 bits of I/O, by simply adding 4 to all the addresses above, e.g.

```
outp(0x0317, 0x9b);        // Program 8255#2 ports A,B,C to be inputs
int a_bits = inp(0x0314);   // Read second 8255 PORT A bits
int b_bits = inp(0x0315);   // Read second 8255 PORT B bits
int c_bits = inp(0x0316);   // Read second 8255 PORT C bits
```

Similarly, if ports are programmed to be outputs, you can WRITE bits to ports A,B,C using outp() instead of inp().

NOTE: According to the 8255 data sheet, the Control Register can ONLY be written to, and not read! In practice I've noticed 82C55 chips let you read the Control Register and see the last value set.

The 8255.EXE program by default reads the Control Register to see how to present the I/O data, unless the user specified the Control Register value byte as the optional argument after the base address, e.g.

```
8255.EXE 0310 9b
-------- ---- --
    |     |   |
    |     |   Control Register value optional argument
    |     |
    |     Base address optional argument
    |
    Runs the 8255 program, usually in
    the \OPCS\BIN directory.
```

SEE ALSO

CIODIO24(DOCS) - Docs on the 8255 based 24 channel Digital I/O card
CIODIO48(DOCS) - Docs on the 8255 based 48 channel Digital I/O card

ORIGIN

Gregory Ercolano, Topanga, California 04/12/00

NAME

NextStep - NextStep micro stepping drives

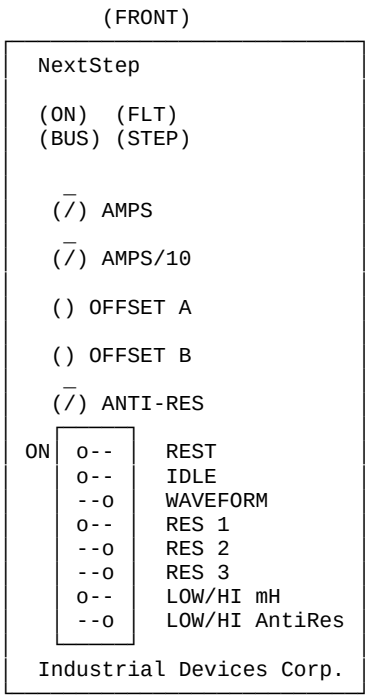
NEXTSTEP STATUS LAMPS

These drives have four status lamps on the front at the top/left:

```
   "ON"  - Grn=Legal power, Yel=Shutdown, Red=Overtemp
   "FLT" - Yel=Interlock unconnected, Red=Motor Short
   "BUS" - Yel(Solid)=Overvoltage, Yel(Flash)=Regen, Red=UnderVoltage
  "STEP" - Grn=Incoming Steps (CW), Yel=Incoming Steps (CCW)
```

NEXTSTEP DIP SWITCHES

The DIP switches along the front of the drive set the power and microstepping resolution:



Drive Resolution (Standard)	Switches RES1	RES2	RES3
5,000	OFF	OFF	ON
10,000	ON	OFF	ON
18,000	OFF	ON	ON
20,000	ON	ON	ON
25,000	OFF	OFF	OFF
25,400	ON	OFF	OFF
36,000	OFF	ON	OFF
50,000	ON	ON	OFF

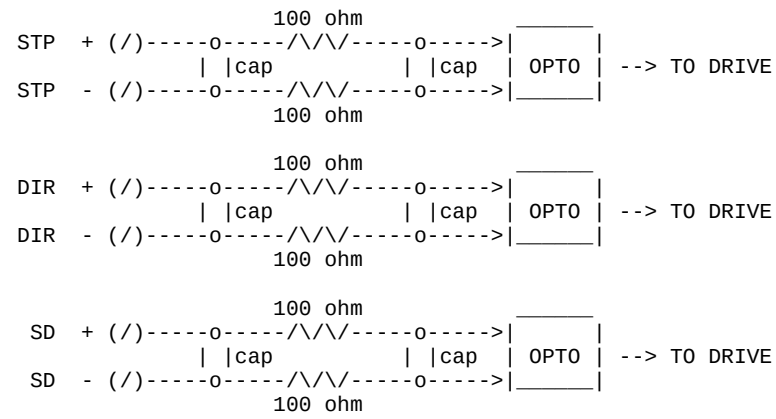
As an example, with SteppersOnline "M0-63-sized" 4.2 amp motors, these settings worked fine:

```
                  REST: ON          AMPS: "2"  \__ together=2.5 amp
                  IDLE: ON       AMPS/10: "5"  /   (low pow setting)
             WAVEFORM: OFF      ANTI-RES: "8"
         LOW/HIGH mH: ON
    LOW/HIGH AntiRes: OFF
```

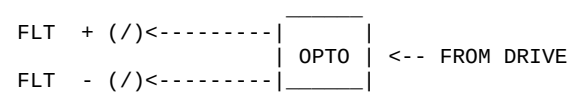
NEXTSTEP STEP/DIR WIRING

There are 3 optically isolated low voltage inputs: STP/DIR/SD
There's one optically isolated low voltage output: FLT

The three inputs are optocoupler inputs with two 100 ohm
resistors in series as shown here:



The one low voltage output is basically the output transistor
of the optocoupler, when the drive indicates a fault condition.



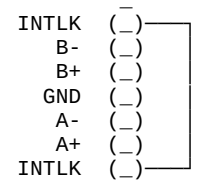
The input ratings are:
- > 5-15mA @5VDC
- > Setup Time=250nS (min on/off time)

The output rating:
- > 50mA max, up to 30VDC.
- > Normally conducting.
- > Opens on fault.

The optocouplers are "HCPL-2631", so you can look up the datasheet
for the electrical particulars.

NEXTSTEP MOTOR WIRING

It is important that the top and bottom terminal screws labeled "INTLK"
are wired *together*. Without this, the output amplifier is turned off
(unlocking the motor) and the "FLT" light will be on. e.g.



The only other wiring needed is to the motor:

- o The B-/B+ to one coil
- o The A-/A+ to other coil

If you swap the wires for the A pairs, it changes the motor direction.

If the motor wire has a shield, it can be connected to the GND pin.

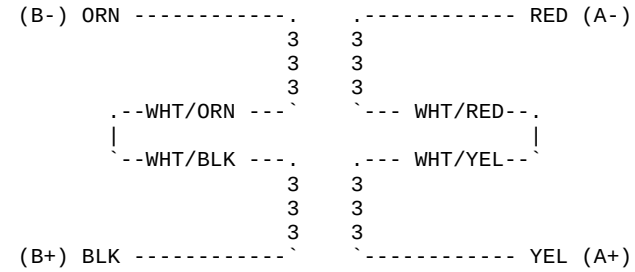
For 4 wire motors, it's simple and you can't really go wrong;
either you get CW, CCW, or no motor turning. It's not possible
to "short anything out" if you miswire the four motor coil wires
to the four A+/A-/B+/B- terminals.

For 6 wire motors, you can choose between SERIES or PARALLEL wiring;

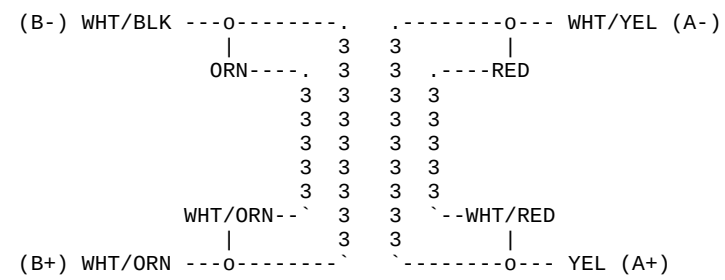
- o SERIES uses lower power and generates lower heat
- o PARALLEL uses more power and generates more heat

In the following diagram, the '3's are the motor coils,
and the wire colors are for the IDC motors depicted in the manual.
See the manual for more information.

For SERIES wiring (low pow/low heat):



For PARALLEL wiring (hi pow/hi heat):



AUTHOR

Greg Ercolano / erco@seriss.com / May 14,2024

NAME

A800 - Seriss Corp. A800 stepper motor control card

DESCRIPTION

The A800 card is a "short slot" ISA card for the IBM PC that can generate steps/direction pulse streams to control up to 8 stepper motors at once.

The card uses two PIC chips to manage the stepper pulse generation. The PIC's firmware and MS-DOS driver "A800DRV.COM" source code are open source and available from:

https://github.com/erco77/a800-opcs-pic-asm

OPCS communicates with the A800 card by way of the MS-DOS device driver "A800DRV.COM", which provides a standard low level interface to the card that OPCS can make use of to run the motors efficiently.

The A800DRV.COM driver must be loaded *before* running the OPCS software. This can be installed either by the AUTOEXEC.BAT, or by a separate batch script that invokes OPCS.

If the A800 card's jumpers are default (BaseAddr=300 and IRQ=5), then you can install the driver with just:

a800drv

CONFIGURING THE BASEADDR AND IRQ

In OPCS K1.xx, the a800 card did not exist and is not supported.

In OPCS K2.00 through K2.09, the base address is configured in OPCSDEFS.OPC with the 'baseaddr' command. IRQ not configurable.

In OPCS K2.10 and up, the A800DRV.COM driver allows both the base address and IRQ to be configured on the command line. The default would be:

```
a800drv -b300 -i5          <-- Sets base address=0300h, IRQ=5
           |    |
           |    IRQ=5
          Base Addr=300
```

..and if your A800 jumpers are set differently, then specify matching values accordingly. e.g. if the card's jumpers are set to BaseAddr=340 and IRQ=6, then start the driver with:

a800drv -b340 -i6

To list the A800DRV driver's options, run 'a800drv -help'. If it does not show a list of options, then it is an older version that does not support command line options.

TECHNICAL SYNOPSIS

When the software wants to move a motor, it provides 8 separate 12 bit velocity values, one per motor channel. And 107 of these velocity values are sent per second to the card using the hardware interrupt on IRQ 5.

Currently only 8 bits of the 12bit value are used for motor speeds. i.e. the lowest velocity is 1 (107 Hz) and the highest velocity is 255 (27,285 Hz). Values above 255 are clipped by the hardware, as the PIC chips are limited by their speed. The high bit (0x8000) is the motor direction bit; 0=forward, 1=reverse.

The software has to keep up with this transmission rate, otherwise it will lose track of the motor positions. The A800DRV.COM device driver provides a 64k ring buffer for the motor velocities that OPCS updates in real time while the motors are running.

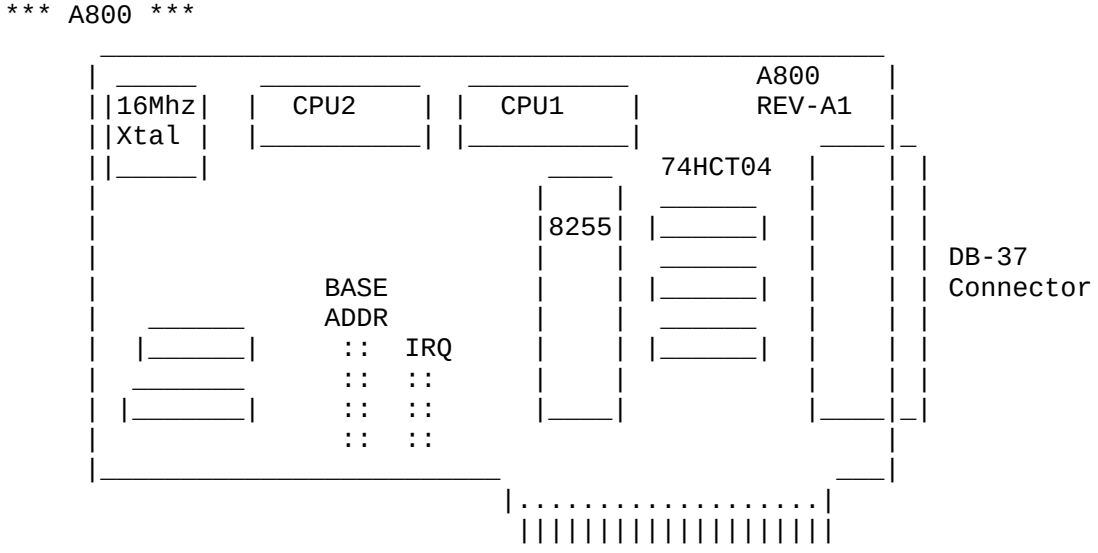
The OPCS software and A800DRV.COM use INT 99h to intercommunicate, providing the address of the ring buffer, and start/stop commands.

The A800 card generates 107 interrupts per second to the A800DRV.COM driver, each interrupt feeds 8 velocities from the tail of the ring buffer to the A800 card, and increments the tail's index to point to the next 8 values in the ring buffer. Meanwhile, the OPCS software feeds velocities into the head of the ring buffer, always keeping ahead of the tail. If the tail catches up to the head prematurely, this causes a SYNC FAULT error, which should never happen unless something is wrong with the computer.

OPCS A800 CARD

==========

This card controls 8 axes and is a half sized IBM PC ISA card.
For complete info on this card, see: http://seriss.com/opcs/a800



DB-37 Connector (similar to Kuper):

PIN#	SIGNAL	PIN	SIGNAL
1	N/C	20	+5VDC
2	STEP A	21	DIR A
3	STEP B	22	DIR B
4	STEP C	23	DIR C
5	STEP D	24	DIR D
6	STEP E	25	DIR E
7	STEP F	26	DIR F
8	STEP G	27	DIR G
9	STEP H	28	DIR H
10	N/C	29	N/C
11	N/C	30	N/C
12	N/C	31	N/C
13	N/C	32	N/C
14	N/C	33	N/C
15	N/C	34	N/C
16	N/C	35	N/C
17	N/C	36	N/C
18	N/C	37	N/C
19	GND(*)		

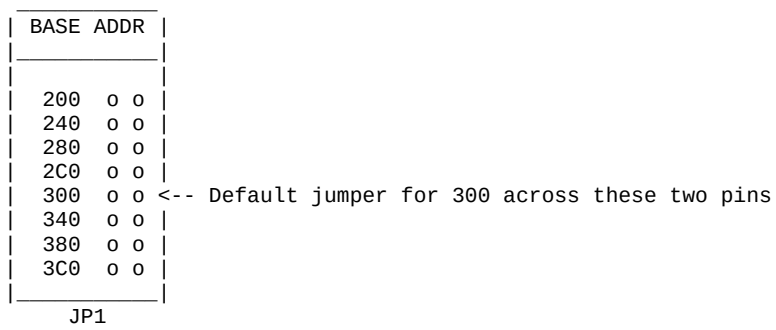
(*) = JP3 configures DB37 Pin#19:
"+5" - Makes Pin #19 +5 VDC
"GND" - Makes Pin #19 GND (default)

NOTE: When fitted with 74LS07 chips, outputs are OPEN COLLECTOR TTL.

When those chips are replaced with 74ALS1034N, outputs swing a full +5/GND and are CMOS/TTL compatible.

BASE ADDRESS (JP1)

Closeup of the 'BASE ADDRESS' jumpers (JP1), which sets the base address of the 8255 chip's I/O port registers:



A800 Base Address Jumpers

Always defer to the board's labeling (if any), as the board designs may have changed since this document's writing (May 2020).

DEFAULTS:
This board has labels for the BASE ADDRESS and IRQs:
"300" is the default base address (5th pair of pins from top jumpered).
"IRQ5" is the default IRQ (4th pair of pins from top jumpered).

DB-37 OUTPUT SIGNALS

The STEPS output are normally high (+5) during idle, and fall low (GND) to pulse the motor a single step.

The outputs for DIR (direction) are logic hi (+5) for forward, and logic low (GND) for reverse.

The output signals can either be CMOS hi/low levels, or can be "open collector" (where logic 'hi' is 'open', and logic low is gnd). Which it is depends on the chips installed in the three chip positions to the left of the DB-37 connector on the A800 board:

```
74HCT04 -- CMOS high/low levels (default)
74LS07  -- Open Collector
```

For controlling the modern DM542 and FMD27400 motor drivers, the 74HCT04 chips are recommended in these positions.

For Centent and Gecko drives, traditionally 74LS07 chips were used, but will probably also work with the 74HCT04's.

While both chips work on all drives, analysis with an oscilloscope monitoring the stepper drive inputs may reveal one chip is better than the other for noise reduction. With 6' cables, 74HCT04 seems the best choice.

Always defer to the board's silk screen labeling, as the board designs may change since this document's writing (May 2020).

HISTORY

Greg Ercolano designed this card in May/June 2020, and the driver software, A800DRV.COM. This card uses "PIC chips", which are programmed with firmware written in the processor's native assembly language for speed and consistent timing for generating the steps and direction motor signals.

SEE ALSO

RTMC16(DOCS) - notes on the Kuper Controls RTMC16 motor control card
RTMC48(DOCS) - notes on the Kuper Controls RTMC48 motor control card
8255(DOCS) - how to control 8255 based digital I/O cards
KUPER(DOCS) - documentation on the kuper card connectors
SD-800(DOCS) - 8 channel stepper distribution board (simplify DB-37 wiring)
SD-1600(DOCS) - 16 channel stepper distribution board (simplify DB-37 wiring)

ORIGIN

Gregory Ercolano, Alhambra, California 06/01/20

NAME

ascii - the ASCII character set encoded in octal, decimal, and hex

DESCRIPTION

ASCII is the American Standard Code for Information Interchange. It is a 7-bit code. Many 8-bit codes (such as ISO 8859-1, the Linux default character set) contain ASCII as their lower half. The international counterpart of ASCII is known as ISO 646.

The following table contains the 128 ASCII characters.

C program '\X' escapes are noted.

Oct	Dec	Hex	Char	Oct	Dec	Hex	Char
000	0	00	NUL '\0'	100	64	40	@
001	1	01	SOH (start of heading)	101	65	41	A
002	2	02	STX (start of text)	102	66	42	B
003	3	03	ETX (end of text)	103	67	43	C
004	4	04	EOT (end of transmission)	104	68	44	D
005	5	05	ENQ (enquiry)	105	69	45	E
006	6	06	ACK (acknowledge)	106	70	46	F
007	7	07	BEL '\a' (bell)	107	71	47	G
010	8	08	BS '\b' (backspace)	110	72	48	H
011	9	09	HT '\t' (horizontal tab)	111	73	49	I
012	10	0A	LF '\n' (new line)	112	74	4A	J
013	11	0B	VT '\v' (vertical tab)	113	75	4B	K
014	12	0C	FF '\f' (form feed)	114	76	4C	L
015	13	0D	CR '\r' (carriage ret)	115	77	4D	M
016	14	0E	SO (shift out)	116	78	4E	N
017	15	0F	SI (shift in)	117	79	4F	O
020	16	10	DLE (data link escape)	120	80	50	P
021	17	11	DC1 (device control 1)	121	81	51	Q
022	18	12	DC2 (device control 2)	122	82	52	R
023	19	13	DC3 (device control 3)	123	83	53	S
024	20	14	DC4 (device control 4)	124	84	54	T
025	21	15	NAK (negative ack.)	125	85	55	U
026	22	16	SYN (synchronous idle)	126	86	56	V
027	23	17	ETB (end of trans. blk)	127	87	57	W
030	24	18	CAN (cancel)	130	88	58	X
031	25	19	EM (end of medium)	131	89	59	Y
032	26	1A	SUB (substitute)	132	90	5A	Z
033	27	1B	ESC (escape)	133	91	5B	[
034	28	1C	FS (file separator)	134	92	5C	\
035	29	1D	GS (group separator)	135	93	5D	]
036	30	1E	RS (record separator)	136	94	5E	^
037	31	1F	US (unit separator)	137	95	5F	_
040	32	20	SPACE	140	96	60	`
041	33	21	!	141	97	61	a
042	34	22	"	142	98	62	b
043	35	23	#	143	99	63	c
044	36	24	$	144	100	64	d
045	37	25	%	145	101	65	e
046	38	26	&	146	102	66	f
047	39	27	?	147	103	67	g
050	40	28	(	150	104	68	h
051	41	29	)	151	105	69	i
052	42	2A	*	152	106	6A	j
053	43	2B	+	153	107	6B	k
054	44	2C	,	154	108	6C	l
055	45	2D	-	155	109	6D	m
056	46	2E	.	156	110	6E	n
057	47	2F	/	157	111	6F	o
060	48	30	0	160	112	70	p
061	49	31	1	161	113	71	q
062	50	32	2	162	114	72	r

063	51	33	3	163	115	73	s
064	52	34	4	164	116	74	t
065	53	35	5	165	117	75	u
066	54	36	6	166	118	76	v
067	55	37	7	167	119	77	w
070	56	38	8	170	120	78	x
071	57	39	9	171	121	79	y
072	58	3A	:	172	122	7A	z
073	59	3B	;	173	123	7B	{
074	60	3C	<	174	124	7C	|
075	61	3D	=	175	125	7D	}
076	62	3E	>	176	126	7E	~
077	63	3F	?	177	127	7F	DEL

Here's the full 256 byte IBM PC character set (ISO-8859, CP-47):

Oct	Dec	Hex	Char	Oct	Dec	Hex	Char
000	000	00	?	200	128	80	Ç
001	001	01	☺	201	129	81	ü
002	002	02	☻	202	130	82	é
003	003	03	♥	203	131	83	â
004	004	04	♦	204	132	84	ä
005	005	05	♣	205	133	85	à
006	006	06	♠	206	134	86	å
007	007	07	<BELL>	207	135	87	ç
010	008	08	<BACKSPACE>	210	136	88	ê
011	009	09	<TAB>	211	137	89	ë
012	010	0a	<LF>	212	138	8a	è
013	011	0b	<VTAB>	213	139	8b	ï
014	012	0c	<FF>	214	140	8c	î
015	013	0d	<CR>	215	141	8d	ì
016	014	0e	♫	216	142	8e	Ä
017	015	0f	☼	217	143	8f	Å
020	016	10	►	220	144	90	É
021	017	11	◄	221	145	91	æ
022	018	12	↕	222	146	92	Æ
023	019	13	‼	223	147	93	ô
024	020	14	¶	224	148	94	ö
025	021	15	§	225	149	95	ò
026	022	16	▬	226	150	96	û
027	023	17	↨	227	151	97	ù
030	024	18	↑	230	152	98	ÿ
031	025	19	↓	231	153	99	Ö
032	026	1a	<EOF>	232	154	9a	Ü
033	027	1b	<ESC>	233	155	9b	¢
034	028	1c	∟	234	156	9c	£
035	029	1d	↔	235	157	9d	¥
036	030	1e	▲	236	158	9e	₧
037	031	1f	▼	237	159	9f	ƒ
040	032	20		240	160	a0	á
041	033	21	!	241	161	a1	í
042	034	22	"	242	162	a2	ó
043	035	23	#	243	163	a3	ú
044	036	24	$	244	164	a4	ñ
045	037	25	%	245	165	a5	Ñ
046	038	26	&	246	166	a6	ª
047	039	27	'	247	167	a7	º
050	040	28	(	250	168	a8	¿
051	041	29	)	251	169	a9	⌐
052	042	2a	*	252	170	aa	¬
053	043	2b	+	253	171	ab	½
054	044	2c	,	254	172	ac	¼
055	045	2d	-	255	173	ad	¡
056	046	2e	.	256	174	ae	«
057	047	2f	/	257	175	af	»
060	048	30	0	260	176	b0	░
061	049	31	1	261	177	b1	▒
062	050	32	2	262	178	b2	▓
063	051	33	3	263	179	b3	│
064	052	34	4	264	180	b4	┤
065	053	35	5	265	181	b5	╡
066	054	36	6	266	182	b6	╢
067	055	37	7	267	183	b7	╖
070	056	38	8	270	184	b8	╕
071	057	39	9	271	185	b9	╣

072	058	3a	:	272	186	ba	‖
073	059	3b	;	273	187	bb	╗
074	060	3c	<	274	188	bc	╝
075	061	3d	=	275	189	bd	╜
076	062	3e	>	276	190	be	╛
077	063	3f	?	277	191	bf	┐
100	064	40	@	300	192	c0	└
101	065	41	A	301	193	c1	┴
102	066	42	B	302	194	c2	┬
103	067	43	C	303	195	c3	├
104	068	44	D	304	196	c4	─
105	069	45	E	305	197	c5	┼
106	070	46	F	306	198	c6	╞
107	071	47	G	307	199	c7	╟
110	072	48	H	310	200	c8	╚
111	073	49	I	311	201	c9	╔
112	074	4a	J	312	202	ca	╩
113	075	4b	K	313	203	cb	╦
114	076	4c	L	314	204	cc	╠
115	077	4d	M	315	205	cd	═
116	078	4e	N	316	206	ce	╬
117	079	4f	O	317	207	cf	╧
120	080	50	P	320	208	d0	╨
121	081	51	Q	321	209	d1	╤
122	082	52	R	322	210	d2	╥
123	083	53	S	323	211	d3	╙
124	084	54	T	324	212	d4	╘
125	085	55	U	325	213	d5	╒
126	086	56	V	326	214	d6	╓
127	087	57	W	327	215	d7	╫
130	088	58	X	330	216	d8	╪
131	089	59	Y	331	217	d9	┘
132	090	5a	Z	332	218	da	┌
133	091	5b	[	333	219	db	█
134	092	5c	\	334	220	dc	▄
135	093	5d	]	335	221	dd	▌
136	094	5e	^	336	222	de	▐
137	095	5f	_	337	223	df	▀
140	096	60	`	340	224	e0	α
141	097	61	a	341	225	e1	ß
142	098	62	b	342	226	e2	Γ
143	099	63	c	343	227	e3	π
144	100	64	d	344	228	e4	Σ
145	101	65	e	345	229	e5	σ
146	102	66	f	346	230	e6	µ
147	103	67	g	347	231	e7	τ
150	104	68	h	350	232	e8	Φ
151	105	69	i	351	233	e9	Θ
152	106	6a	j	352	234	ea	Ω
153	107	6b	k	353	235	eb	δ
154	108	6c	l	354	236	ec	∞
155	109	6d	m	355	237	ed	φ
156	110	6e	n	356	238	ee	ε
157	111	6f	o	357	239	ef	∩
160	112	70	p	360	240	f0	≡
161	113	71	q	361	241	f1	±
162	114	72	r	362	242	f2	≥
163	115	73	s	363	243	f3	≤
164	116	74	t	364	244	f4	⌠
165	117	75	u	365	245	f5	⌡
166	118	76	v	366	246	f6	÷
167	119	77	w	367	247	f7	≈
170	120	78	x	370	248	f8	°
171	121	79	y	371	249	f9	∙
172	122	7a	z	372	250	fa	·
173	123	7b	{	373	251	fb	√
174	124	7c	|	374	252	fc	ⁿ
175	125	7d	}	375	253	fd	²
176	126	7e	~	376	254	fe	■
177	127	7f	⌂	377	255	ff	

Tables

For convenience, let us give more compact tables in hex and decimal.

```
  2 3 4 5 6 7       30 40 50 60 70 80 90 100 110 120
 ------------      ------------------------------
0:   0 @ P ` p     0:    (  2  <  F  P  Z  d   n   x
1: ! 1 A Q a q     1:    )  3  =  G  Q  [  e   o   y
2: " 2 B R b r     2:    *  4  >  H  R  \  f   p   z
3: # 3 C S c s     3: !  +  5  ?  I  S  ]  g   q   {
4: $ 4 D T d t     4: "  ,  6  @  J  T  ^  h   r   |
5: % 5 E U e u     5: #  -  7  A  K  U  _  i   s   }
6: & 6 F V f v     6: $  .  8  B  L  V  `  j   t   ~
7: ? 7 G W g w     7: %  /  9  C  M  W  a  k   u  DEL
8: ( 8 H X h x     8: &  0  :  D  N  X  b  l   v
9: ) 9 I Y i y     9: ?  1  ;  E  O  Y  c  m   w
A: * : J Z j z
B: + ; K [ k {
C: , < L \ l |
D: - = M ] m }
E: . > N ^ n ~
F: / ? O _ o DEL
```

NOTES

History

An ascii manual page appeared in Version 7 of AT&T UNIX.

On older terminals, the underscore code is displayed as a left arrow, called backarrow, the caret is displayed as an up-arrow and the vertical bar has a hole in the middle.

Uppercase and lowercase characters differ by just one bit and the ASCII character 2 differs from the double quote by just one bit, too. That made it much easier to encode characters mechanically or with a non-microcontroller-based electronic keyboard and that pairing was found on old teletypes.

The ASCII standard was published by the United States of America Standards Institute (USASI) in 1968.

SEE ALSO

iso_8859-1(7), iso_8859-10(7), iso_8859-13(7), iso_8859-14(7), iso_8859-15(7),iso_8859-16(7), iso_8859-2(7), iso_8859-3(7), iso_8859-4(7), iso_8859-5(7), iso_8859-6(7), iso_8859-7(7), iso_8859-8(7), iso_8859-9(7)

COLOPHON

This page is part of release 3.22 of the Linux man-pages project. A description of the project, and information about reporting bugs, can be found at http://www.kernel.org/doc/man-pages/.

NAME

centent - centent stepper motor drive wiring

CENTENT MOTOR WIRING

These pinouts are straight out of the Centent CNO143 manual. Quothe the Centent docs:

> "One motor winding connects to T3 and T4, while the other winding connects to T5 and T6. The step motor drive will operate 4, 6, and 8 wire motors. The 6 and 8 wire motors may be wired in either a series (low power) or parallel (hi power) configuration. For 8 wire motors, follow the manufacturer's hookup diagrams for series or parallel operation." (See below)

In the following table, T3, T4, T5 and T6 refer to the numbered terminals on the drive. Colors indicate the wire colors that come off the motor shown.

*** SERIES WIRING - CNO-143 ***

Manufacturer	T3	T4	T5	T6
Superior Electric	GRN/WHT	GRN	RED	RED/WHT
Rapidsyn	GRN/WHT	GRN	RED	RED/WHT
IMC	GRN/WHT	GRN	RED	RED/WHT
Sigma	YEL	RED	BLK	ORN
Oriental Motor	BLU	RED	BLK	GRN
Portescap	YEL/WHT	RED	ORN/WHT	BRN
Bodine	YEL	RED	BRN	ORN
Digital Motor	YEL	RED	BLK	ORN
Warner	RED	YEL	ORN	BRN
Japan Servo	YEL	GRN	BLU	RED

*** PARALLEL WIRING - CNO-143 ***

Manufacturer	T3	T4	T5	T6
Superior Electric	WHT	GRN	RED	BLK
Rapidsyn	WHT	GRN	RED	BLK
IMC	WHT	GRN	RED	BLK
Sigma	RED/YEL	RED	ORN/BLK	ORN
Oriental Motor	BLU	WHT	YEL	GRN
Portescap	RED/WHT	RED	ORN/WHT	ORN
Bodine	RED/WHT	RED	ORN/WHT	ORN
Digital Motor	RED/WHT	RED	BLK	BLK/WHT
Warner	RED	WHT	ORN	BLK
Japan Servo	WHT	GRN	BLU	WHT

FIELD WIRING NOTES

These are actual wiring diagrams I've used in the field. These are NOT from the Centent docs; use at own risk. Although these work, double check manufacturer's recommended wiring; see SLOSYN(DOCS) man page ('**man slosyn**') for info.

```
        ***       SERIES (LO POWER)      ***
        ***    CENTENT CNO-143 WIRING    ***

    Superior Ele.   Centent         Superior Ele.
    8 wire motor    CNO-143         6 wire motor
    ------------    -------         ------------
    wht/grn         3               wht/grn
    grn             4               grn
    red             5               red
    wht/red         6               wht/red
(*) wht & blk       NOT CONNECTED   blk
(*) wht/blk & orn   NOT CONNECTED   wht

        ***      PARALLEL (HI POWER)     ***
        ***    CENTENT CNO-143 WIRING    ***

    Superior Ele.   Centent         Superior Ele.
    8 wire motor    CNO-143         6 wire motor
    ------------    -------         ------------
(*) wht/blk & orn   3               wht
    grn             4               grn
    red             5               red
(*) wht & blk       6               blk
    wht/grn         x               wht/grn
    wht/red         x               wht/red

(*) 8 WIRE NOTE: With 8 wire connections, WHT and BLK are
tied together, but are not connected to the drive. Also,
WHT/BLK and ORN are similarly tied together.
```

The following tables are for Superior Electric motors, and pretty much re-iterate the Centent docs, albeit clearer to my eyes:

FULL WINDING (LO POWER)

PHASE	CENTENT DRIVE TERMINAL	MOTOR WIRE COLOR
"A"	3	GRN/WHT
"B"	4	GRN
"C"	5	RED/WHT
"D"	6	RED

HALF WIRING (HI POWER)

PHASE	CENTENT DRIVE TERMINAL	MOTOR WIRE COLOR
"A"	3	WHT
"B"	4	GRN
"C"	5	BLK
"D"	6	RED

CIODIO24(DOCS) Optical Printer Control System CIODIO24(DOCS)

NAME

CIO-DIO24 - Docs for the CIO-DIO24 (8255 based) Digital I/O Card

DESCRIPTION

The CIO-DIO24 is an ISA 8255 based Digital I/O card, available from http://www.computerboards.com/

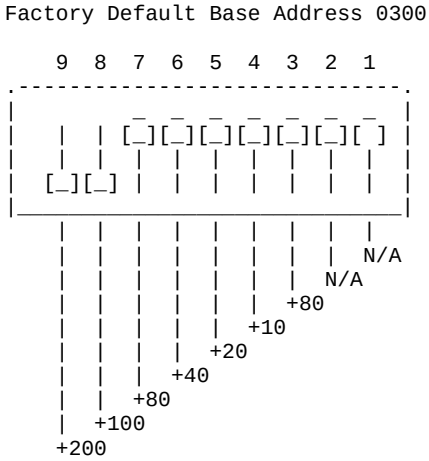
This board is an ISA 8255 Digital I/O board with selectable port base address.

CIO-DIO24 PINOUT AND SWITCH SETTINGS

```
          ---------------------------------------
                        CIO-DIO24
                        Male DB37
          ---------------------------------------

          (LEFT ROW)                    (RIGHT ROW)
          Pin Description               Pin Description
          --- -----------               --- -----------
           19 GND                |\      37 A0
           18 +5                 |  \    36 A1
           17 GND                |   |   35 A2
           16 +12                |   |   34 A3
           15 GND                |   |   33 A4
           14 -12                |   |   32 A5
           13 GND                |   |   31 A6
           12 -5                 |   |   30 A7
           11 GND                |   |   29 B0
           10 B0                 |   |   28 B1
            9 B1                 |   |   27 B2
            8 B2                 |   |   26 B3
            7 B3                 |   |   25 B4
            6 B4                 |   |   24 B5
            5 B5                 |   |   23 B6
            4 B6                 |   |   22 B7
            3 B7                 |   |   21 GND
            2 IR/ENA             |  /    20 +5
            1 IR/IN              |/
```

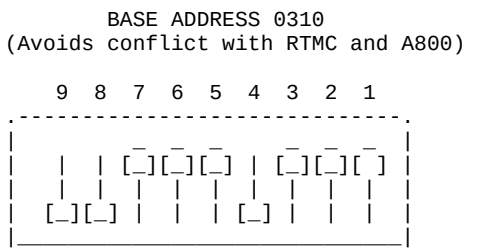
CIO-DIO24 DIP SWITCH (PORT BASE ADDRESS SET)

Factory default setting is '300' (hex) for the dip switches; 9 & 8 down. Switches are active when in the 'down' position.



NOTE: Switches are active or "ON" in the 'down' position on these boards.

To avoid conflict with stepper pulse boards like the Kuper RTMC16, RTMC48, and the OPCS A800 boards, it's best to use 0310 instead, e.g.



So if the board's base address is 0310, then the address of the 8255 chip is assigned this way:

Description	Base Address	Offset
Port A	0310	BaseAddr+0
Port B	0311	BaseAddr+1
Port C	0312	BaseAddr+2
I/O Control Register	0313	BaseAddr+3

See below for a description of the possible values for the "I/O Control Register".

8255 CHIP DOCS

The following are the docs for the onboard 8255 chip.

8255 PORTS

The 8255 family of chips (82C55, etc) are usually one chip solutions to getting 24 bits of programmable digital I/O.

Typically, the chip is configured at a base I/O address, such as 0310.

There are four ports (base+0, base+1, base+2 and base+3) that are used to control the chip:

base+0 - PORT #A
base+1 - PORT #B
base+2 - PORT #C
base+3 - CONTROL REGISTER

8255 PROGRAMMING

The "control register" controls the I/O direction of the 3 ports, and breaks out as follows:

```
Bit   Description
---   -------------------------------------------------
 0    Port C (low 4 bits): 1=input, 0=output
 1    Port B (all 8 bits): 1=input, 0=output
 2    Mode selection: 0=MODE#0, 1=MODE#1
 3    Port C (hi 4 bits): 1=input, 0=output
 4    Port A (all 8 bits): 1=input, 0=output
   _
 5  |_ 00 = MODE#0 (basic I/O)
 6  _|  01 = MODE#1 (strobed I/O)
        1x = MODE#2 (bidirectional bus)

 7    Mode set flag (1=active, 0=normal)
```

No initialization is 'required' to achieve 24 bits of input; it's the default. During reset, all ports are programmed to be inputs.

The control register must be programmed before doing any I/O with the three ports A,B and C. Example values for the Control Register:

```
          0x80    - A,B,C outputs
          0x9b    - A,B,C inputs
          0x92    - A+B input, C=output
```

8255 PROGRAMMING EXAMPLES

Here's a C programming example that shows how to setup the 8255 such that A+B are inputs, and C is outputs:

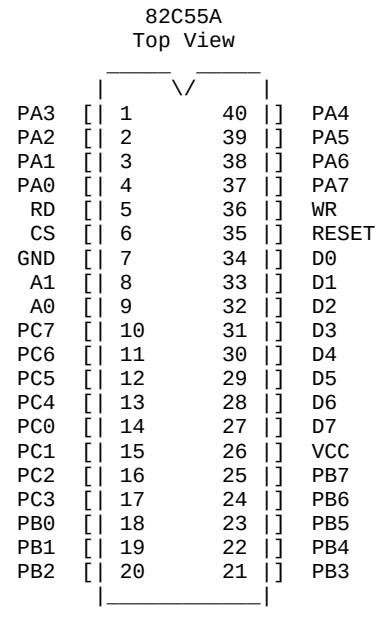
```
/* INITIALIZATION */
base = 0x310;
out(base+3), 0x92;      /* A+B=in, C=out */
/* READ/WRITE */
if ( inp(base+0) & 0x01 )         /* read A */
    printf("Bit #1 set on port A\n");
else
    printf("Bit #1 clear on port A\n");
/* SET PORT C, BIT #1 */
out(base+2, inp(base+2) | 0x01 ); /* write C */
```

This example shows similar 8255 programming example in the OPCS system's HOMEDEFS.HOM file, used by the HOME program:

```
# homedefs.hom
start init_8255
{
    # INITIALIZATION
    outport 0313 92        # A+B=in, C=out

    # READ/WRITE
    portset? 0310 01       # test if bit 1 set on Port A
    {
        print "Bit #1 set on Port A"
    }
    portclr? 0310 01       # test if bit 1 clear on Port A
    {
        print "Bit #1 clear on Port A"
    }

    # SET PORT C, BIT #1
    setbit 0312 01
}
end init_8255
```

8255 CHIP PINOUT

The 82C55A is most commonly found in a 40 pin DIP.



8255 PORT MONITOR PROGRAM

The OPCS software comes with 8255.exe which can monitor the real time status of the 8255's I/O ports.

Run '8255.exe' with the hex base address as the first argument:

```
C:\OPCS\WORK> 8255.EXE 0310
              -------- ----
                  |     |
                  |     Base address argument
                  |
                  Runs the 8255 program, usually in
                  the \OPCS\BIN directory.
```

This tool can be downloaded from https://seriss.com/opcs/ftp/ and the source code on githib at https://github.com/erco77/8255-dos/

CONTROL REGISTER: BASE+3

The control register is a single 8 bit port whose byte defines the I/O direction of all three 8 bit I/O ports A,B,C, and the mode for that I/O, modes 0,1 and 2 are supported by the 8255 chips.

The control register for each 8255 should be programmed on boot or OPCS startup to define whether ports A, B and C are inputs or outputs.

Initializing the control register can be done using the "home" program, creating a special entry in the HOMEDEFS.HOM file to initialize the port (see example below), and then invoking "**home 8255_init**" during the AUTOEXEC.BAT or via the OPCSDEFS.OPC setup file that's run when OPCS starts.

Example 8255 init using HOMEDEFS.HOM, where the base address for the 8255 is 0200:

1. Create a new entry called 'init_8255' in the HOMEDEFS.HOM file:

```
# Initialize the 8255 CIO/DIO board
start init_8255
{
    outport 0203 9b    # program the 8255 Control Register
}
end init_8255
```

   Replace 0203 with the base address+3 of your 8255 board, and replace '9b' with the Control Register value appropriate for your setup, as it defines which ports are input vs. output.

   See the table below "Control Register - "Mode 0" Operation" for a list of all the possible I/O configuration control register values.

2. Create an entry in either the AUTOEXEC.BAT file, or the batch file you use to start OPCS, using:

   home init_8255

   ..which programs the 8255's control register based on the above HOMEDEFS.HOM file entry.

Note that individual bits can NOT BE programmed separately;

> Port A can be programmed to either be "all in" or "all out"
> Same for Port B
> Port C, the LOWER and UPPER 4 bits can be controlled separately for Input or Output.

I strongly suggest referring to the 8255 and/or 82C55 data sheets for authoritative information about how to program the 8255 for I/O. Secondarily the manual for the CIO-DIO48 ISA card.

What follows below is my reduction of those docs. (erco@seriss.com)

Control Register - "Mode 0" Operation

This document ONLY covers "Mode 0" of the 8255, which is simple "real time I/O" for all ports. Mode 1 (Strobed I/O) and Mode 2 (Bi-Directional Bus) are not covered here. For more info on those features, refer to the 8255 data sheet.

```
            Group A       Group B
            I/O Ctrl      I/O Ctrl
              _____         _____
             /     \       /     \
 ___________ _______ ___ _______ _________________ _________________ ___
|   |   |   | GROUP |M1 | GROUP |   GROUP A       |   GROUP B       |   |
|MS |M3 |M2 |   A   |   |   B   |_________________|_________________|   |
|___|___|___|_______|___|_______|       |         |       |         |Ctl|
|D7 |D6 |D5 |D4 |D3 |D2 |D1 |D0 |PORT A|PORT C(HI)|PORT B|PORT C(LO)|Reg|
|---|---|---|---|---|---|---|---|------|----------|------|----------|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | OUT  |  OUT     | OUT  |  OUT     |80 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | OUT  |  OUT     | OUT  |  In      |81 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | OUT  |  OUT     |  In  |  OUT     |82 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | OUT  |  OUT     |  In  |  In      |83 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | OUT  |   In     | OUT  |  OUT     |88 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | OUT  |   In     | OUT  |  In      |89 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | OUT  |   In     |  In  |  OUT     |8A |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | OUT  |   In     |  In  |  In      |8B |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |  In  |  OUT     | OUT  |  OUT     |90 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |  In  |  OUT     | OUT  |  In      |91 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |  In  |  OUT     |  In  |  OUT     |92 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |  In  |  OUT     |  In  |  In      |93 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |  In  |   In     | OUT  |  OUT     |98 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |  In  |   In     | OUT  |  In      |99 |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |  In  |   In     |  In  |  OUT     |9A |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |  In  |   In     |  In  |  In      |9B |
 ---------------------------------------------------------------------------
/|\ /|\ /|\         /|\
 |   |   |           |
 |   |   |           All zeroes (for "Mode 0")
 |   |   All zeroes (for "Mode 0")
 |   All zeroes (for "Mode 0")
 All ones (for "Mode 0")

 MS = Mode Set bit (hi bit of control register), must be 1 for "Mode 0"
 M3,M2,M1 = must all be 0 for "Mode 0".
```

Here are all the possible 8 bit Control Register values (in HEX) to program Ports A,B,C in "Mode 0" for all possible Input/Output combinations:

```
 Control Register          (HI 4)           (LO 4)
    Value (hex)    PORT A   PORT C   PORT B   PORT C
 ---------------  -------  -------  -------  -------
        80          Out     Out       Out     Out  <- Port A,B,C=Out
        81          Out     Out       Out     In
        82          Out     Out       In      Out
        83          Out     Out       In      In
        88          Out     In        Out     Out
        89          Out     In        Out     In
        8A          Out     In        In      Out
        8B          Out     In        In      In
        90          In      Out       Out     Out
        91          In      Out       Out     In
        92          In      Out       In      Out   <- Port A,B=In, C=Out
        93          In      Out       In      In
        98          In      In        Out     Out
        99          In      In        Out     In
        9A          In      In        In      Out
        9B          In      In        In      In    <- Port A,B,C=In
```

(The above table is from the CIO-DIO48 ISA board manual for the 82C55)

Put simply, the bit mask for the control register I/O control for Mode 0 for ports A,B,C is:

```
   Data Bit  Mask  Description
   --------  ----  -----------
      D0     01h   Port C(LO)
      D1     02h   Port B
      D3     08h   Port C(HI)
      D4     10h   Port A

   NOTE: "D2" bit is skipped because it's the M1 mode bit
```

So if the base address is 0310, one can program ports A,B and C
of the first 24 bits of I/O to all be inputs using the C code:

```
outp(0x0313, 0x9b);         // Program ports A,B,C to be inputs
```

And to read the input bits for ports A,B,C:

```
int a_bits = inp(0x0310);   // Read PORT A bits
int b_bits = inp(0x0311);   // Read PORT B bits
int c_bits = inp(0x0312);   // Read PORT C bits
```

The same can be done for the second 24 bits of I/O, by simply adding 4
to all the addresses above, e.g.

```
outp(0x0317, 0x9b);         // Program 8255#2 ports A,B,C to be inputs
int a_bits = inp(0x0314);   // Read second 8255 PORT A bits
int b_bits = inp(0x0315);   // Read second 8255 PORT B bits
int c_bits = inp(0x0316);   // Read second 8255 PORT C bits
```

Similarly, if ports are programmed to be outputs, you can WRITE bits
to ports A,B,C using outp() instead of inp().

NOTE: According to the 8255 data sheet, the Control Register can ONLY be
written to, and not read! In practice I've noticed 82C55 chips
let you read the Control Register and see the last value set.

The 8255.EXE program by default reads the Control Register
to see how to present the I/O data, unless the user specified
the Control Register value byte as the optional argument
after the base address, e.g.

```
8255.EXE 0310 9b
-------- ---- --
    |     |   |
    |     |   Control Register value optional argument
    |     |
    |     Base address optional argument
    |
    Runs the 8255 program, usually in
    the \OPCS\BIN directory.
```

ORIGIN

Gregory Ercolano, Topanga, California 04/12/00

CIODIO48(DOCS) Optical Printer Control System CIODIO48(DOCS)

NAME

CIO-DIO48 - Docs for the CIO-DIO48 (dual 8255 based) Digital I/O Card

DESCRIPTION

The CIO-DIO48 is an ISA based Digital I/O card available from: http://www.computerboards.com/

This board contains two 8255 chips that control 24 bits each for a total of 48 separate I/O signals.

What follows are the CIO-DIO48 connector pinout, dip switch settings, and programming considerations.

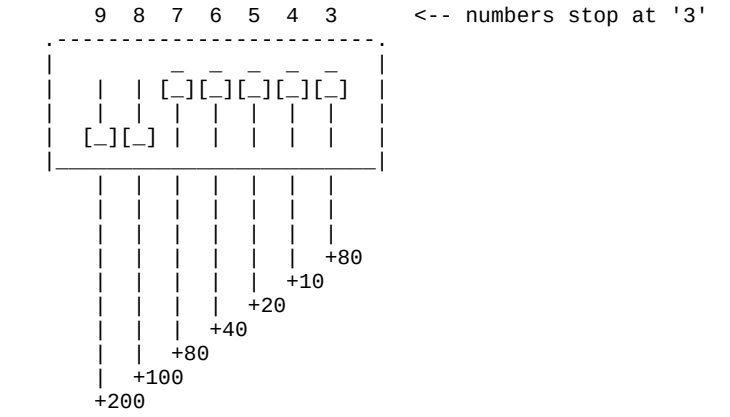
BASE ADDRESS DIP SWITCH

The "base address" of the CIO-DIO48 board determines what ports software uses to read, write, and configure the two 8255 chips on the CIO-DIO48.

Unlike "Plug and Play" PCI boards, the older ISA boards had to have their addresses configured manually by DIP switches.

Factory default setting is '300' (hex) for the dip switches; 9 & 8 down. Switches are active when in the 'down' position.

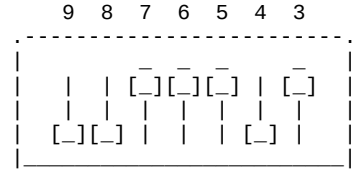
Factory Default Base Address 0300



NOTE: Switches are active or "ON" in the 'down' position on these boards.

To avoid conflict with stepper pulse boards like the Kuper RTMC16, RTMC48, and the OPCS A800 boards, it's best to use 0310 instead, e.g.

BASE ADDRESS 0310
(Avoids conflict with RTMC and A800)



So if the board's base address is 0310, then the address of the 8255 chip is assigned this way:

```
                        Base
Description             Address  Offset
--------------------    -------  ----------    __
                Port A  0310     BaseAddr+0      |
                Port B  0311     BaseAddr+1      |_ 8255 #1
                Port C  0312     BaseAddr+2      |
I/O Control Register    0313     BaseAddr+3    __|
                                               __
                Port A  0314     BaseAddr+4      |
                Port B  0315     BaseAddr+5      |_ 8255 #2
                Port C  0316     BaseAddr+6      |
I/O Control Register    0317     BaseAddr+7    __|
```

The I/O Control Register's values are described below, and define whether ports A/B/C are Inputs or Outputs.

50 PIN IDC CONNECTOR

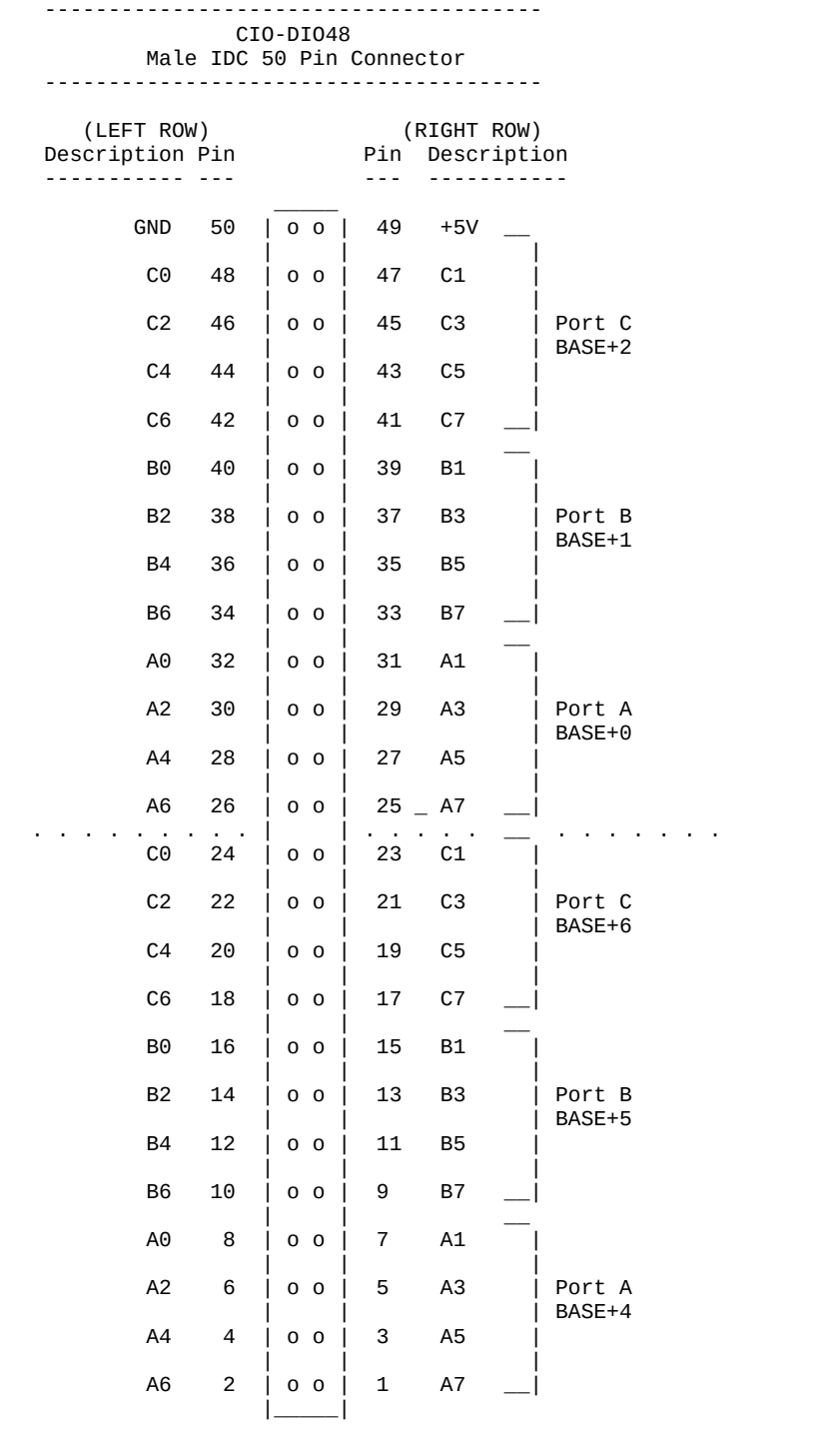
The CIO-DIO48 is really two 8255 chips on a single ISA board, each capable of managing 24 I/O bits for a total of 48.

The 24 I/O bits of each 8255 is broken up into three separate 8 bit ports "A", "B" and "C".

The CIO-DIO48 card's 50 pin IDC connector pinout is:

> Pins 01 - 24: First group of 24 bits
> Pins 25 - 48: Second group of 24 bits
> Pins 49(+5) and 50(GND) are power supply pins.

Here's the complete pinout on the male connector, looking down at the connector's pins from the component side of the board:



If the base address is 0310 (hex), then:

> I/O Port A is 0310 for pins 1 thru 8
> I/O Port B is 0311 for pins 9 thru 16
> I/O Port C is 0312 for pins 17 thru 24
> Control register for these three ports is 0313

LOW 24 bits

> I/O Port A is 0314 for pins 25 thru 32
> I/O Port B is 0315 for pins 33 thru 40
> I/O Port C is 0316 for pins 41 thru 48
> Control register for these three ports is 0317

HIGH 24 bits

8255 CHIP DOCS

The following are the docs for the onboard 8255 chip.

8255 PORTS

The 8255 family of chips (82C55, etc) are usually one chip solutions to getting 24 bits of programmable digital I/O.

Typically, the chip is configured at a base I/O address, such as 0310.

There are four ports (base+0, base+1, base+2 and base+3) that are used to control the chip:

```
base+0    - PORT #A
base+1    - PORT #B
base+2    - PORT #C
base+3    - CONTROL REGISTER
```

8255 PROGRAMMING

The "control register" controls the I/O direction of the 3 ports, and breaks out as follows:

```
Bit   Description
---   -------------------------------------------------
 0    Port C (low 4 bits): 1=input, 0=output
 1    Port B (all 8 bits): 1=input, 0=output
 2    Mode selection: 0=MODE#0, 1=MODE#1
 3    Port C (hi 4 bits): 1=input, 0=output
 4    Port A (all 8 bits): 1=input, 0=output
   _
 5  |_ 00 = MODE#0 (basic I/O)
 6  _|  01 = MODE#1 (strobed I/O)
        1x = MODE#2 (bidirectional bus)

 7    Mode set flag (1=active, 0=normal)
```

No initialization is 'required' to achieve 24 bits of input; it's the default. During reset, all ports are programmed to be inputs.

The control register must be programmed before doing any I/O with the three ports A,B and C. Example values for the Control Register:

```
0x80    - A,B,C outputs
0x9b    - A,B,C inputs
0x92    - A+B input, C=output
```

8255 PROGRAMMING EXAMPLES

Here's a C programming example that shows how to setup the 8255 such that A+B are inputs, and C is outputs:

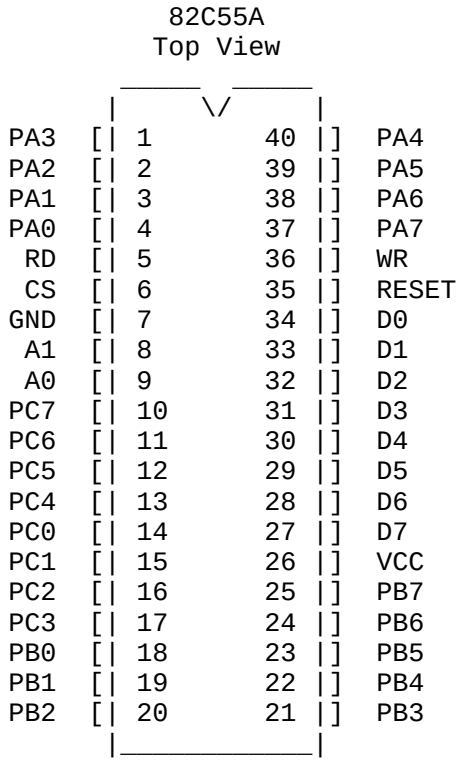
```
/* INITIALIZATION */
base = 0x310;
out(base+3), 0x92;      /* A+B=in, C=out */
/* READ/WRITE */
if ( inp(base+0) & 0x01 )         /* read A */
    printf("Bit #1 set on port A\n");
else
    printf("Bit #1 clear on port A\n");
/* SET PORT C, BIT #1 */
out(base+2, inp(base+2) | 0x01 ); /* write C */
```

This example shows similar 8255 programming example
in the OPCS system's HOMEDEFS.HOM file, used by the HOME program:

```
# homedefs.hom
start init_8255
{
    # INITIALIZATION
    outport 0313 92        # A+B=in, C=out

    # READ/WRITE
    portset? 0310 01       # test if bit 1 set on Port A
    {
        print "Bit #1 set on Port A"
    }
    portclr? 0310 01       # test if bit 1 clear on Port A
    {
        print "Bit #1 clear on Port A"
    }

    # SET PORT C, BIT #1
    setbit 0312 01
}
end init_8255
```

8255 CHIP PINOUT

The 82C55A is most commonly found in a 40 pin DIP.



8255 PORT MONITOR PROGRAM

The OPCS software comes with 8255.exe which can monitor the
real time status of the 8255's I/O ports.

Run '8255.exe' with the hex base address as the first argument:

```
C:\OPCS\WORK> 8255.EXE 0310
              -------- ----
                  |     |
                  |     Base address argument
                  |
                  Runs the 8255 program, usually in
                  the \OPCS\BIN directory.
```

This tool can be downloaded from https://seriss.com/opcs/ftp/
and the source code on githib at https://github.com/erco77/8255-dos/

CONTROL REGISTER: BASE+3

The control register is a single 8 bit port whose byte defines the I/O direction of all three 8 bit I/O ports A,B,C, and the mode for that I/O, modes 0,1 and 2 are supported by the 8255 chips.

The control register for each 8255 should be programmed on boot or OPCS startup to define whether ports A, B and C are inputs or outputs.

Initializing the control register can be done using the "home" program, creating a special entry in the HOMEDEFS.HOM file to initialize the port (see example below), and then invoking "**home 8255_init**" during the AUTOEXEC.BAT or via the OPCSDEFS.OPC setup file that's run when OPCS starts.

Example 8255 init using HOMEDEFS.HOM, where the base address for the 8255 is 0200:

1. Create a new entry called 'init_8255' in the HOMEDEFS.HOM file:

```
# Initialize the 8255 CIO/DIO board
start init_8255
{
    outport 0203 9b    # program the 8255 Control Register
}
end init_8255
```

   Replace 0203 with the base address+3 of your 8255 board, and replace '9b' with the Control Register value appropriate for your setup, as it defines which ports are input vs. output.

   See the table below "Control Register - "Mode 0" Operation" for a list of all the possible I/O configuration control register values.

2. Create an entry in either the AUTOEXEC.BAT file, or the batch file you use to start OPCS, using:

   home init_8255

   ..which programs the 8255's control register based on the above HOMEDEFS.HOM file entry.

Note that individual bits can NOT BE programmed separately;

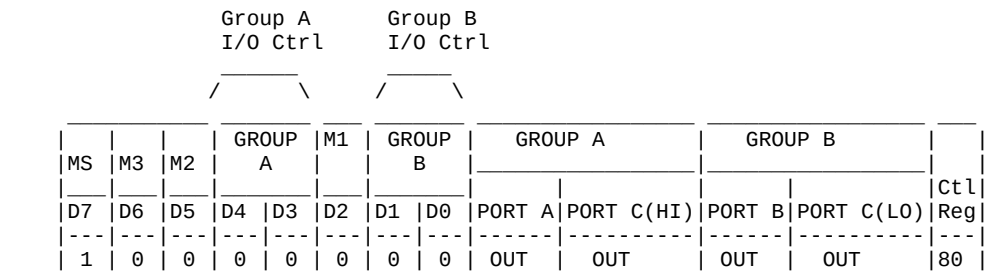
> Port A can be programmed to either be "all in" or "all out"
> Same for Port B
> Port C, the LOWER and UPPER 4 bits can be controlled separately for Input or Output.

I strongly suggest referring to the 8255 and/or 82C55 data sheets for authoritative information about how to program the 8255 for I/O. Secondarily the manual for the CIO-DIO48 ISA card.

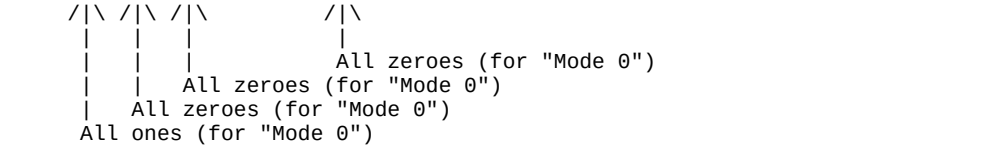
What follows below is my reduction of those docs. (erco@seriss.com)

Control Register - "Mode 0" Operation

This document ONLY covers "Mode 0" of the 8255, which is simple "real time I/O" for all ports. Mode 1 (Strobed I/O) and Mode 2 (Bi-Directional Bus) are not covered here. For more info on those features, refer to the 8255 data sheet.



MS	M3	M2	GROUP A	M1	GROUP B	GROUP A		GROUP B		Ctl
D7	D6	D5	D4	D3	D2	D1	D0	PORT A	PORT C(HI)	PORT B	PORT C(LO)	Reg
1	0	0	0	0	0	0	0	OUT	OUT	OUT	OUT	80
1	0	0	0	0	0	0	1	OUT	OUT	OUT	In	81
1	0	0	0	0	0	1	0	OUT	OUT	In	OUT	82
1	0	0	0	0	0	1	1	OUT	OUT	In	In	83
1	0	0	0	0	1	0	0	OUT	In	OUT	OUT	88
1	0	0	0	0	1	0	1	OUT	In	OUT	In	89
1	0	0	0	0	1	1	0	OUT	In	In	OUT	8A
1	0	0	0	0	1	1	1	OUT	In	In	In	8B
1	0	0	0	1	0	0	0	In	OUT	OUT	OUT	90
1	0	0	0	1	0	0	1	In	OUT	OUT	In	91
1	0	0	0	1	0	1	0	In	OUT	In	OUT	92
1	0	0	0	1	0	1	1	In	OUT	In	In	93
1	0	0	0	1	1	0	0	In	In	OUT	OUT	98
1	0	0	0	1	1	0	1	In	In	OUT	In	99
1	0	0	0	1	1	1	0	In	In	In	OUT	9A
1	0	0	0	1	1	1	1	In	In	In	In	9B



MS = Mode Set bit (hi bit of control register), must be 1 for "Mode 0"
M3,M2,M1 = must all be 0 for "Mode 0".

Here are all the possible 8 bit Control Register values (in HEX) to program Ports A,B,C in "Mode 0" for all possible Input/Output combinations:

Control Register Value (hex)	PORT A	(HI 4) PORT C	PORT B	(LO 4) PORT C	
80	Out	Out	Out	Out	<- Port A,B,C=Out
81	Out	Out	Out	In	
82	Out	Out	In	Out	
83	Out	Out	In	In	
88	Out	In	Out	Out	
89	Out	In	Out	In	
8A	Out	In	In	Out	
8B	Out	In	In	In	
90	In	Out	Out	Out	
91	In	Out	Out	In	
92	In	Out	In	Out	<- Port A,B=In, C=Out
93	In	Out	In	In	
98	In	In	Out	Out	
99	In	In	Out	In	
9A	In	In	In	Out	
9B	In	In	In	In	<- Port A,B,C=In

(The above table is from the CIO-DIO48 ISA board manual for the 82C55)

Put simply, the bit mask for the control register I/O control for Mode 0 for ports A,B,C is:

Data Bit	Mask	Description
D0	01h	Port C(LO)
D1	02h	Port B
D3	08h	Port C(HI)
D4	10h	Port A

NOTE: "D2" bit is skipped because it's the M1 mode bit

So if the base address is 0310, one can program ports A,B and C of the first 24 bits of I/O to all be inputs using the C code:

```
outp(0x0313, 0x9b);         // Program ports A,B,C to be inputs
```

And to read the input bits for ports A,B,C:

```
int a_bits = inp(0x0310);   // Read PORT A bits
int b_bits = inp(0x0311);   // Read PORT B bits
int c_bits = inp(0x0312);   // Read PORT C bits
```

The same can be done for the second 24 bits of I/O, by simply adding 4 to all the addresses above, e.g.

```
outp(0x0317, 0x9b);         // Program 8255#2 ports A,B,C to be inputs
int a_bits = inp(0x0314);   // Read second 8255 PORT A bits
int b_bits = inp(0x0315);   // Read second 8255 PORT B bits
int c_bits = inp(0x0316);   // Read second 8255 PORT C bits
```

Similarly, if ports are programmed to be outputs, you can WRITE bits to ports A,B,C using outp() instead of inp().

NOTE: According to the 8255 data sheet, the Control Register can ONLY be written to, and not read! In practice I've noticed 82C55 chips let you read the Control Register and see the last value set.

The 8255.EXE program by default reads the Control Register to see how to present the I/O data, unless the user specified the Control Register value byte as the optional argument after the base address, e.g.

```
    8255.EXE 0310 9b
    -------- ---- --
        |     |   |
        |     |   Control Register value optional argument
        |     |
        |     Base address optional argument
        |
        Runs the 8255 program, usually in
        the \OPCS\BIN directory.
```

ORIGIN

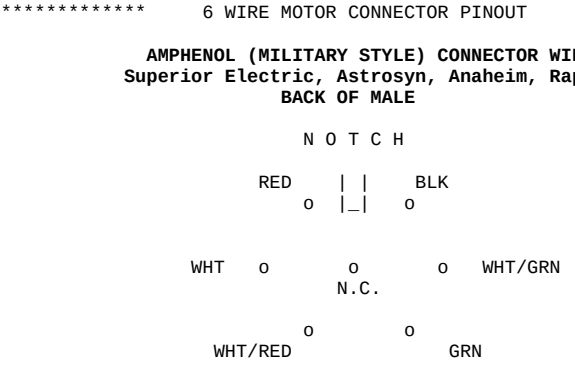
Gregory Ercolano, Alhambra, California 08/02/23

NAME

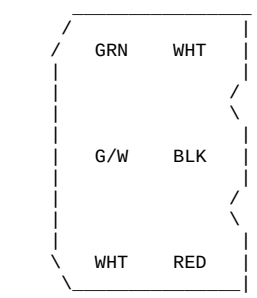
connector - misc connector wiring

```
***************          SUPERIOR ELECTRIC            ************************
***************     6 WIRE MOTOR CONNECTOR PINOUT     ************************
```

AMPHENOL (MILITARY STYLE) CONNECTOR WIRING
Superior Electric, Astrosyn, Anaheim, Rapidsyn
BACK OF MALE



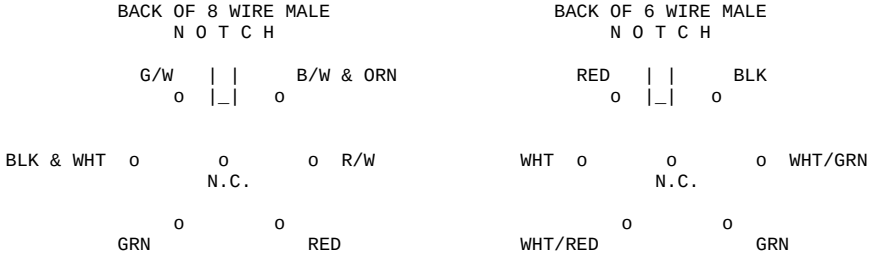
6 PIN NYLON (MEDIUM DUTY) RADIO SHACK CONNECTOR
Superior Electric, Astrosyn, Anaheim, Rapidsyn
BACK OF MALE



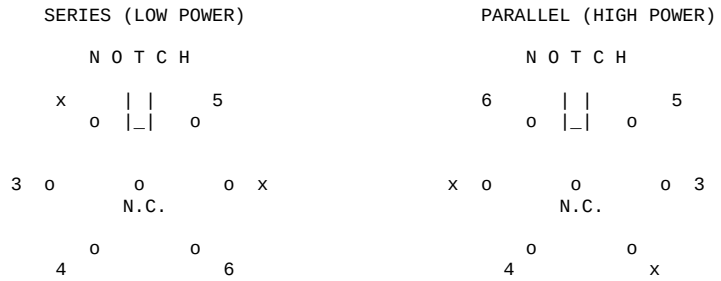
Back of Male

```
***************          SUPERIOR ELECTRIC            ************************
***************     8 WIRE MOTOR CONNECTOR PINOUT     ************************
```

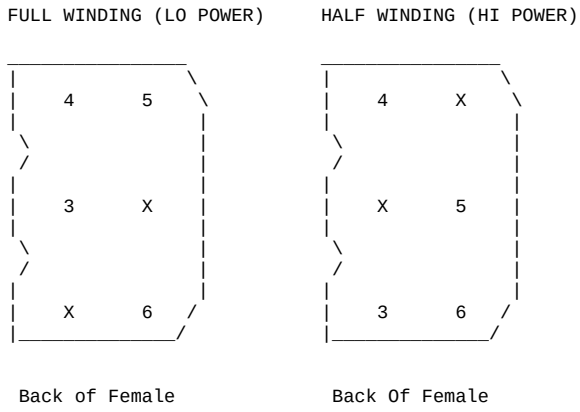
AMPHENOL (MILITARY STYLE) CONNECTOR WIRING
Superior Electric, Astrosyn, Anaheim, Rapidsyn



AMPHENOL (MILITARY STYLE) CONNECTOR WIRING
Superior Electric, Astrosyn, Anaheim, Rapidsyn
Centent terminals shown
BACK OF FEMALE



6 PIN NYLON (MEDIUM DUTY) RADIO SHACK CONNECTOR
BACK OF FEMALE CONNECTORS
Centent Terminals Shown



EASE(TOOLS) Optical Printer Control Systems EASE(TOOLS)

NAME

ease - create ease in / out positions

SYNOPSIS

ease file chans sfrm efrm spos epos eifrm eofrm [etype]

file - the name of the position file
chans - channels to effect. A channel can be any letter a thru l. Can be ranges like 'a-d', or comma delimited lists like 'a,d,e' or combinations like 'a,d-j'.
sfrm - starting frame of move
efrm - ending frame of move
spos - starting position
epos - ending position
eifrm - ease in # frames
eofrm - ease out # frames
etype - 1.0=linear ease, .25=hard ease(.75=default)

EXAMPLE

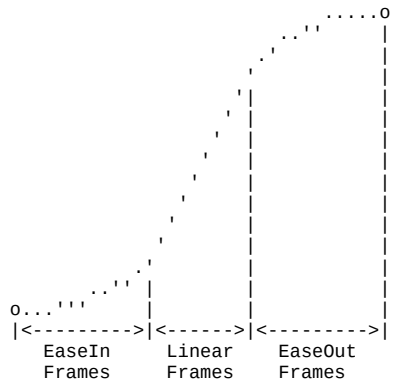
ease foo.pos f 1 50 1000 2000 10 10

```
------- ---- ---- --------- -----
File    Chan Frms Positions Ease
```

DESCRIPTION

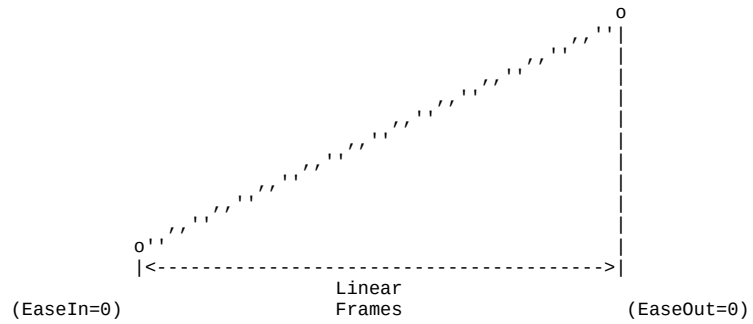
ease creates moves (columns of numbers) in ascii OPCS position files. Given a range of frames and positions, ease(OPCS) will create smooth moves between two points.

The move created between the positions has 3 parts:



Exponential curves are used to achieve the ease-in and ease-outs. The linear section is a straight linear interpolation is created between the two ease curves.

If the easein/easeout values are *zero*, then no ease curves are created, and an accurate straight linear move is made between the start and end positions. e.g.:



POSITION FILES

Position files are ASCII files that contain 12 positions per line. Each line represents a single frame of positions. Each vertical column of numbers represent a single channel. The left most column is the 'a' channel, the right most column is the 'l' channel.

A position file:

> May contain comment lines that start with '#' or ';'

> Will be limited to 12 channels in width

> Can only contain values that are long integers

> Each column of numbers must be separated by white space

> Blank lines are ignored

> Can be any length. Position files are never loaded entirely into memory, so they can be extremely long files.

DEBUG

To enable debugging, '**set EASEDEBUG=1**' before running ease.

AUTHOR

Greg Ercolano, Venice California 1998

NAME

gecko - gecko stepper motor drive wiring (201, 201X..)

GECKO MOTOR WIRING

Excerpts from Gecko 201 manual and other sources for actual wire colors.

Motors rated phase current: 0.3 AMP - 7 AMP. Power supply voltage should be between 4x and 20x the motor rated voltage. The current set resistor may be 1/4W, 5%. Current set resistor goes across T11 and T12. Values:

HEATSINK OPTIONAL		HEATSINK REQUIRED	
MOTOR CURRENT	RESISTOR VALUE	MOTOR CURRENT	RESISTOR VALUE
1.0 AMP	7.8K	4.0 AMP	62.7K
1.5 AMP	12.8K	4.5 AMP	84.6K
2.0 AMP	18.8K	5.0 AMP	117.5K
2.5 AMP	26.1K	5.5 AMP	172.3K
3.0 AMP	35.2K	6.0 AMP	282.0K
3.5 AMP	47.0K	6.5 AMP	611.0K
		7.0 AMP	OPEN

Gecko 201 Terminal Arrangement

Terminal#	Description
T1	Supply Ground
T2	Supply Power (+) (+18VDC to +80VDC)
T3	Phase A
T4	Phase B
T5	Phase C
T6	Phase D
T7	Disable windings (when connected to ground T12)
T8	Direction (ground-going signal relative to +5V common)
T9	Step (ground-going signal relative to +5V common)
T10	+5V Common
T11	Current set resistor
T12	Current set resistor (ground)

Use heat sinks for current settings above 3 amps.
Drive should be heatsinked to a piece of aluminum, preferably with fins and a fan to increase heat dissipation and surface area.

GECKO OPTION JUMPERS

```
   1  2  3  4
  .------------.        > Jumper pins 2 and 6 for STANDBY ENABLED
  | o  o  o  o |        > Jumper pins 1 and 5 for STANDBY DISABLED
  | o  o  o  o |        > Leave all pins open for LOW CURRENT RANGE
  `------------'        > Jumper pins 3 and 7 for SIZE 42 MOTOR
    5  6  7  8          > Jumper pins 4 and 8 for MID-BAND DISABLED
                         > Jumper pins 7 and 8 for NORMAL (DEFAULT)
```

GECKO COMPUTER CONNECTIONS

On the Kuper cards and A800 boards, connect STEPS to T9, DIRECTION to T8, and +5v from card (pin 20) to T10, e.g.

Kuper RTMC/A800 Pin	Gecko Drive Terminals
1	(no connection)
2 (A-Step)	T9 for channel A
3 (B-Step)	T9 for channel B
4 (C-Step)	T9 for channel C
: :	:
etc etc	etc
: :	:
20 (+5VDC)	T10 on ALL GECKO DRIVES
21 (A-Dir)	T8 for channel A
22 (B-Dir)	T8 for channel B
23 (C-Dir)	T8 for channel C
: :	:
etc etc	etc
: :	:

GECKO MOTOR CONNECTIONS

In the following table, T3, T4, T5 and T6 refer to the numbered terminals on the drive. Colors indicate the wire colors that come off the motor. NOTE: These are derived from Centent docs, which /should/ be compatible. From what I could find, Gecko doesn't provide diagrams showing wire colors, but they seem to be terminal compatible with the Centent drives (which does).

********* GECKO 201 - SERIES WIRING - 6 WIRE *********

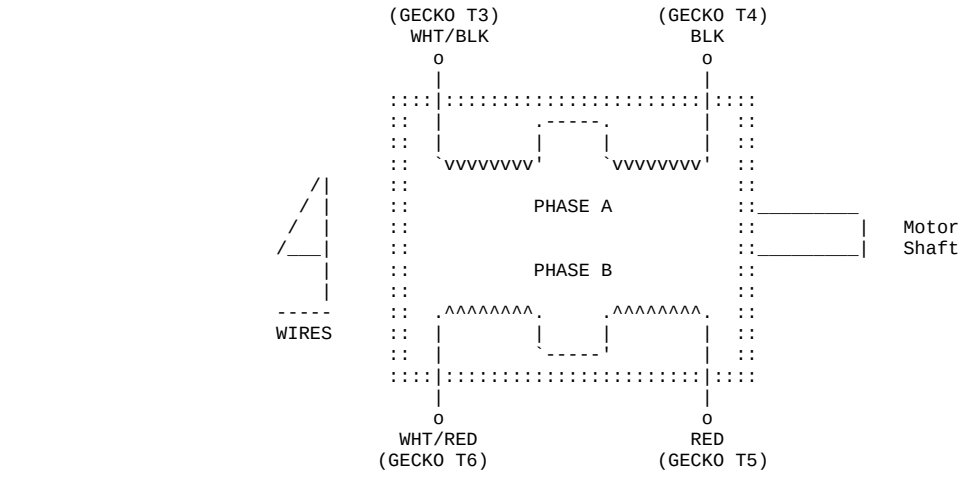
Manufacturer	T3	T4	T5	T6
Superior Electric	GRN/WHT	GRN	RED	RED/WHT
Rapidsyn	GRN/WHT	GRN	RED	RED/WHT
IMC	GRN/WHT	GRN	RED	RED/WHT
Sigma	YEL	RED	BLK	ORN
Oriental Motor	BLU	RED	BLK	GRN
Portescap	YEL/WHT	RED	ORN/WHT	BRN
Bodine	YEL	RED	BRN	ORN
Digital Motor	YEL	RED	BLK	ORN
Warner	RED	YEL	ORN	BRN
Japan Servo	YEL	GRN	BLU	RED

********* GECKO 201 - PARALLEL WIRING - 6 WIRE *********

Manufacturer	T3	T4	T5	T6
Superior Electric	WHT	GRN	RED	BLK
Rapidsyn	WHT	GRN	RED	BLK
IMC	WHT	GRN	RED	BLK
Sigma	RED/YEL	RED	ORN/BLK	ORN
Oriental Motor	BLU	WHT	YEL	GRN
Portescap	RED/WHT	RED	ORN/WHT	ORN
Bodine	RED/WHT	RED	ORN/WHT	ORN
Digital Motor	RED/WHT	RED	BLK	BLK/WHT
Warner	RED	WHT	ORN	BLK
Japan Servo	WHT	GRN	BLU	WHT

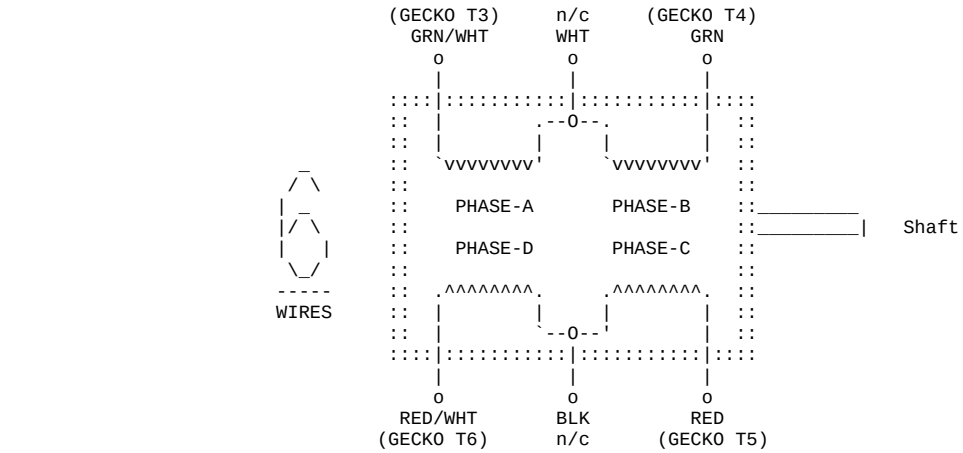
4-WIRE MOTOR WIRING DIAGRAM

The following diagram shows the motor wiring for a 4 wire stepper motor.



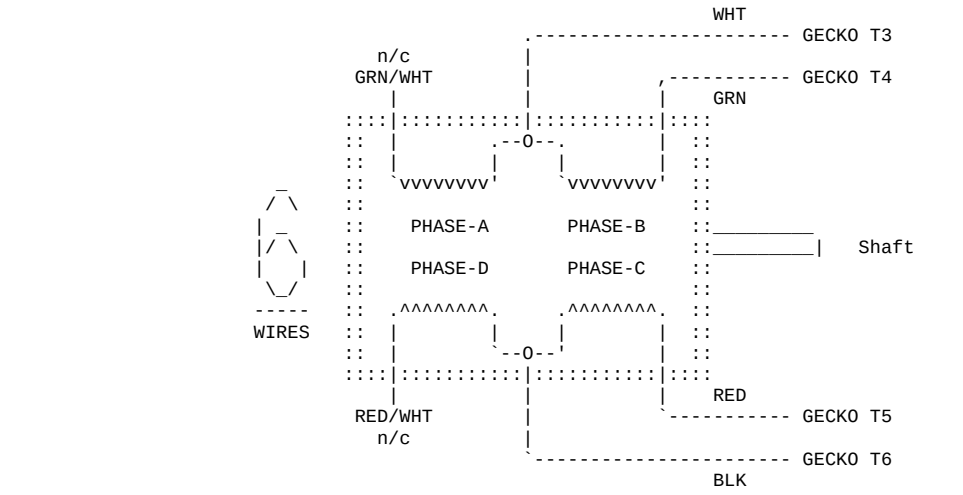
SUPERIOR ELECTRIC MOTOR 6-WIRE - SERIES WIRING DIAGRAM

The following diagram shows the SERIES "low power" motor wiring for for a Superior Electric 6 wire motor.



SUPERIOR ELECTRIC MOTOR 6-WIRE - PARALLEL

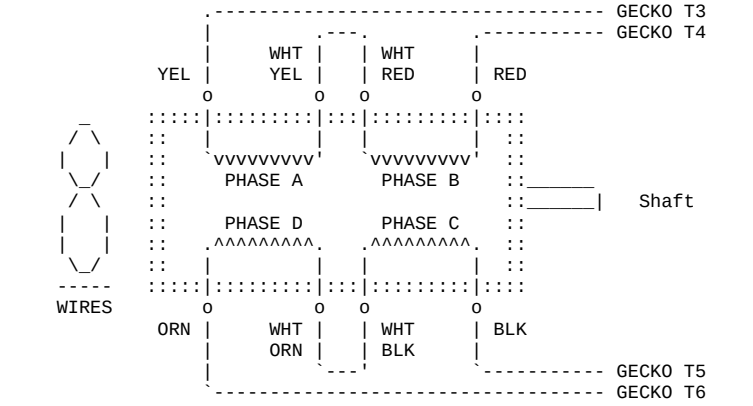
PARALLEL "high power" wiring for a Superior Electric 6 wire motor.



********** **SUPERIOR ELECTRIC MOTOR** **********
********** **8 WIRE - SERIES** **********

SERIES "low power" wiring for a Superior Electric 8 wire motor.

Tie WHT/YEL + WHT/RED together, isolate.

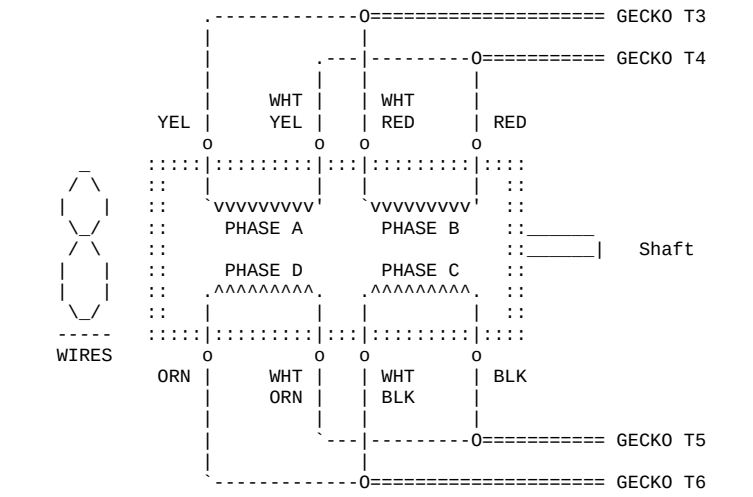


Tie WHT/ORN + WHT/BLK together, isolate.

********** **SUPERIOR ELECTRIC MOTOR** **********
********** **8 WIRE - PARALLEL** **********

PARALLEL "high power" wiring for a Superior Electric 8 wire motor.

Tie YEL + WHT/RED together, connect to T3
Tie RED + WHT/YEL together, connect to T4



Tie ORN + WHT/BLK together, connect to T5
Tie BLK + WHT/ORN together, connect to T6

GR(DOCS) Optical Printer Control Systems GR(DOCS)

NAME

gr - graph moves

SYNOPSIS

gr file chans [sfrm efrm]

```
file  - the name of the position file
chans - channels to show in graph. A channel can be any letter a thru l.
        Can be ranges like 'a-d', or comma delimited lists like
        'a,d,e' or combinations like 'a,d-j'.
sfrm  - starting frame of move
efrm  - ending frame of move
```

If any of the options are not supplied, the user will be prompted.

EXAMPLE

gr foo.pos e-f 1 50

```
------- ----- ----
File    Chans Frames
```

DESCRIPTION

gr is an external .exe tool that runs in DOS, but can be run from within OPCS using either '**! gr foo.pos e-f 1 50**' or gr can be added as an OPCS command using DOSCMD(OPCSDEFS).

gr will graph the columns of numbers in the specified positions file.

EDITING KEYS

```
Left/Right Arrow - Select a different frame (also ^B/^F)
   Tab/Shift-Tab - Jump forward/reverse frames 10% of screen
   Up/Down Arrow - Adjust position +/- 1/20th screen (also ^P/^N)
    Ctrl-Up/Down - Micro-adjust positions +/- 1
               R - Redraw screen
               S - Enter new frame range
         ESC, Q - Quit
            A-L - Select channel A through L
             ^E - Toggles 'noclear' mode, leaves trail of edit positions
             ^M - Modift current position by typing in new value
```

One can use the arrow keys to adjust positions on screen:

HARDWARE

It is assumed your video card is capable of displaying rudimentary 640x350 16 color graphics (EGA/VGA). All graphics cards should support this mode. BIOS mode 16, e.g. INT 10H, AH=0, AL=10H.

AUTHOR

Greg Ercolano, Venice California 1998

HOME(DOCS) Optical Printer Control System HOME(DOCS)

NAME
home - motor homing standalone program

OPTIONS
-v # verbose mode for debugging changes
-d # debug mode
-h # help

USAGE
home [name-of-procedure] ..

EXAMPLES
home # home all axes defined in 'default' procedure
home a b c # call procedures 'a', 'b' and 'c'

DESCRIPTION
'home' is an external .EXE command that runs in DOS, but can be easily available within OPCS by adding '**doscmd home**' to OPCSDEFS.OPC.

'home' is part of the OPCS system, and is used to home the camera, projector, fader, and motion control axes. It can set bits, check for hardware conditions, etc.

'home' has a simple command language that lets one run motors until the status bit of home sensors change, and there's also logical if() conditions that can be used to check hardware bits.

DEFINING HOMING PROCEDURES
Homing procedures are defined this way:

```
start foo
{
    commands..
}
end foo
```

..where 'foo' can be any word or letter that is used to call the procedure, and can be invoked from the command line as:

```
home foo
```

In the case of setting up homing procedures for channels, the procedure names are the channel letter, e.g.:

```
start a
{
    print AERIAL HOME

    # ..code to handle homing the aerial projector..
}
end a
```

..so to home the 'a' channel, one just executes:

```
home a
```

The names used for 'start' and 'end' are whatever you want. By convention, channel letters are used for defining procedures for homing those channels. You can also define your own procedures for doing other things, like homing wedge wheels, and other such things that the OPCSDEFS.OPC file can't do.

The HOMEDEFS.HOM file has two special procedures:

```
start always
{
        # commands that always execute, and are executed *first*;
}
end always

start default
{
        # Commands to execute if user just typed
        # 'home' without any command line arguments.
}
end default
```

When you run 'home' without arguments, the 'default' procedure in the HOMEDEFS.HOM file is executed. So it's common to have that procedure home all the motors, e.g.

```
start default
{
    call a     # home aerial (if there is one)
    call b     # home main
    call c     # home camera
    call d     # home fader
}
end default
```

COMMANDS

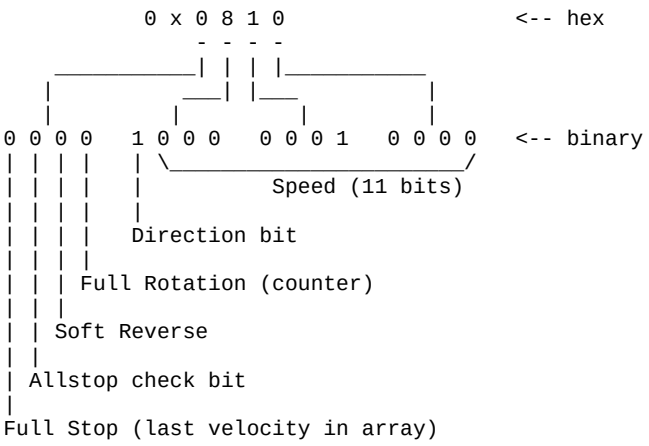
Home commands are put into the HOMEDEFS.HOM file, enclosed in procedures. Here's a list of all the home commands (K2.20/TC):

```
   baseaddr [port]                # Kuper/A800 base address [0300 default]
      call [proc]                 # Call named procedure
     clrbit [port] [bitmask]      # Clear port bit by OR/XORing [bitmask]
     cswait [centisecs]           # Wait so many centiseconds
        end [label]               # Declare the end of a procedure
       exit [code]                # Exit with an error code
      fail?  { commands }         # Run { cmds } if gohome/gochange cmd failed
         go [chan] [dist] [spd]   # Send a motor some distance
   gochange [chan] [maxdist] [spd] # Run until home sensor changes state
     gohome [chan] [maxdist] [spd] # Home a motor
       goto [label]               # goto a label within proc (label-name:)
   homeport [chan] [port] [mask] [test] # Port to test for home condition
    ishome? [chan] { cmds }       # Is a channel at its home position?
    istrip? [chan] { cmds }       # Is a chan's trip switch is tripped?
   nothome? [chan] { cmds }       # Is chan NOT at its home position?
    notrip? [chan] { cmds }       # Is a chan's trip switch NOT tripped?
    outport  [port] [bytevalue]   # Write a byte to a port
      pass?  { commands }         # Run { cmds } if go/gohome/gochange passed
   portset? [port] [mask] { cmds } # Check if port bit set
   portclr? [port] [mask] { cmds } # Check if port bit clear
        ppr [chan] [value]       # Pulses per revolution
      print [string]              # Prints a message to the screen
  printport [port]                # Print the value at a port in hex
        pse                       # Wait for a keypress from the user
      reset [chan] [value]        # Reset counter for [chan] to [value]
sampspersec [val]                 # Vel samples per second:
                                  # RTMC=120.0, A800=107.0, A800(REVB)=120.0
     setbit [port] [bitmask]      # Set port bit by ORing [bitmask]
      start [label]               # Declare beginning of a procedure
                (NOTE: label name 'always' defines procedure ALWAYS parsed)
     system [command]             # Execute a DOS command
   tripport [chan] [port] [mask] [test] # Port to test for trip condition
     xorbit [port] [bitmask]      # Invert port bit by XORing [bitmask]
```

ENVIRONMENT VARIABLES

OPCSDEBUG - if set to '1', 'home' will dump the velocity arrays to stdout after the motors have moved. The output can be redirected to a file for better inspection.

The 16 bit velocity values printed are 16 bit words of the form:



The lower 12 bits (dir + speed) are sent directly to the Kuper card. The upper 4 bits are interpreted by the 'mdrive' resident driver code.

EXAMPLE HOMEDEFS.HOM FILE

```
#########################################################################
# HOME PROGRAM'S DEFINITIONS FILE
# This file contains commands particular only to the 'home' program.
# 'home' is a standalone program that can move the motors, and watch
# for changes in the home sensors.
#########################################################################

# DEFINE PORTS FOR THE USER'S INSTALLATION

start always    # THESE COMMANDS *ALWAYS* PARSED
{
    ### HOME SENSORS
    homeport a 0000 00 00       homeport e 0000 00 00   homeport i 0000 00 00
    homeport b 0000 00 00       homeport f 0000 00 00   homeport j 0000 00 00
    homeport c 0000 00 00       homeport g 0000 00 00   homeport k 0000 00 00
    homeport d 0000 00 00       homeport h 0000 00 00   homeport l 0000 00 00

    ### TRIP SWITCHES
    tripport a 0000 00 01       tripport e 0000 00 01   tripport i 0000 00 01
    tripport b 0000 00 01       tripport f 0000 00 01   tripport j 0000 00 01
    tripport c 0000 00 01       tripport g 0000 00 01   tripport k 0000 00 01
    tripport d 0000 00 01       tripport h 0000 00 01   tripport l 0000 00 01
}
end always

start a
{
    print
    print AERIAL: (DISABLED)
}
end a
```

```
start b
{
    print -n   MAIN:
    ishome? b
    {
        print -n RUNOUT FROM HOME -
        go b -500 .4
        ishome? b
        {
            print
            print *** Cant run out of home position! (are motors off?)
            exit 1
        }
        cswait 20
    }
    print -n SEEKING HOME -
    gohome b 1000 .4
    fail?
    {
        print -n SEEK HOME IN REV DIR -
        cswait 50
        go b -1500 .4   # (arrive at home from same direction)
        cswait 50
        gohome b 2000 .4
        fail?
        {
            print
            print *** Cant find home position! (are motors off?)
            exit 1
        }
    }
    print DONE.
}
end b

start c
{
    print -n CAMERA:
    ishome? c
    {
        print -n RUNOUT FROM HOME -
        go c -500 .4
        ishome? c
        {
            print
            print *** Cant run out of home position! (are motors off?)
            exit 1
        }
        cswait 20
    }
    print -n SEEKING HOME -
    gohome c 1000 .4
    fail?
    {
        print -n SEEK HOME IN REV DIR -
        cswait 50
        go c -1500 .4   # (arrive at home from same direction)
        cswait 50
        gohome c 2000 .4
        fail?
        {
            print
            print *** Cant find home position! (are motors off?)
            exit 1
        }
    }
    print DONE.
}
end c

start d
{
    print -n  FADER:
    ishome? d
    {
        print -n RUNOUT FROM HOME -
```

```
        go d 1500 .11
        ishome? d
        {
            print
            print *** Cant run out of home position! (are motors off?)
            exit 1
        }
        cswait 20
    }
    print -n SEEKING HOME -
    cswait 50
    gohome d -11450 .11
    fail?
    {
        print
        print *** Cant find home position! (are motors off?)
        exit 1
    }
    cswait 10
    go d -1950 .11
    print  DONE.
}
end d

# EXAMPLE OF INITIALIZING A LINEAR AXIS
#    Change the X to a channel letter.
#
start X
{
    print -n  CAM ZOOM:
    ishome? X
    {
        print -n MOVE OFF,
        gochange X -100000 .25
        ishome? X
        {
            print
            print *** Cant find home! (are motors off?)
            exit 1
        }
        # GO A LITTLE FURTHER IN SAME DIRECTION
        go X -400 .25
        cswait 20
    }
    print -n MOVE TOWARD EDGE,
    gochange X 100000 .25
    print DONE.
}
end X

# USED BY JOG(OPCS)
start deenergize
{
    print *** HOMEDEFS.HOM: No 'deenergize' target defined.
}
end deenergize

exit 0
```

ORIGIN

Gregory Ercolano, Topanga, California 05/12/00

NAME

kuper - kuper card documentation

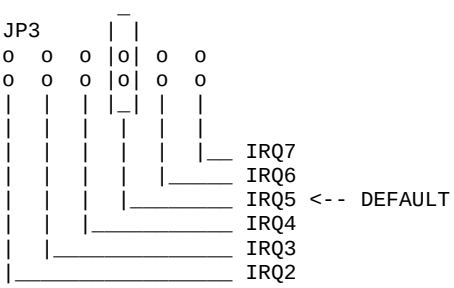
KUPER CONTROL RTMC16 CARD CONNECTOR 'P2'
(DB37S - 37 pin connector)

```
--------------------------------
   1  - (*)    20 - +5VDC
   2  - STEP A 21 - DIR A        DB37S (37 pin connector)
   3  - STEP B 22 - DIR B
   4  - STEP C 23 - DIR C
   5  - STEP D 24 - DIR D
   6  - STEP E 25 - DIR E
   7  - STEP F 26 - DIR F         (*) = Jumper Select GND or +5 with JP5:
   8  - STEP G 27 - DIR G               +5VDC - Short pins 1 & 2 on JP5
   9  - STEP H 28 - DIR H               GND  - Short pins 2 & 3 on JP
   10 - STEP I 29 - DIR I
   11 - STEP J 30 - DIR J
   12 - STEP K 31 - DIR K         NOTE: All outputs are OPEN COLLECTOR
   13 - STEP L 32 - DIR L               TTL. Maximum +5 current draw
   14 - STEP M 33 - DIR M               should not exceed 400 milliamps.
   15 - STEP N 34 - DIR N
   16 - STEP O 35 - DIR O
   17 - STEP P 36 - DIR P
   18 - (*)    37 - +5VDC
   19 - (*)
```

KUPER CONTROL CARD IRQ JUMPER SETTINGS (JP3)
Selects the IRQ used for feeding velocities

```
--------------------------------
```

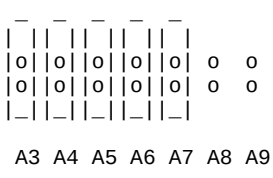
JP3 has a single jumper on one set of pins to set the IRQ;
from left to right, pins set IRQ 2 thru 7, with 5 being the default:



KUPER CONTROL CARD SWITCH SETTINGS 'JP4'
Selects the KuperBase address value
(Jumpers A3-A9)

```
--------------------------------
```

This is the default configuration for 0300h:



Here's the table of the JP4 jumper variations from the
RTMC16 manual, shown in "most likely to work" order:

KuperBase Address	A3	A4	A5	A6	A7	A8	A9
0300	1	1	1	1	1	0	0
0320	1	1	0	1	1	0	0
0320	0	1	0	1	1	0	0
0330	1	0	0	1	1	0	0
0340	1	1	1	0	1	0	0
0280	1	1	1	1	0	1	0
02a0	1	1	0	1	0	1	0
0308	0	1	1	1	1	0	0
0310	1	0	1	1	1	0	0
0318	0	0	1	1	1	0	0

NOTE: 0 = off 1 = on

Factory setting is 0300, and there is usually no need
to modify this setting unless other boards in the machine
are conflicting with this address. Same for the IRQ setting.

KUPER LOGIC CONNECTOR [R1]
Inputs are tied high to +5

PIN	NAME	PORT	MASK (hex)
1	GND	GND	GND
2	out 0	0x306	01
3	out 1	0x306	02
4	out 2	0x306	04
5	out 3	0x306	08
6	out 4	0x306	10
7	out 5	0x306	20
8	out 6	0x306	40
9	out 7	0x306	80
10-12	???	???	??
13	+5	+5	+5
14	GND	GND	GND
15	in 0	0x306	01
16	in 1	0x306	02
17	in 2	0x306	04
18	in 3	0x306	08
19	in 4	0x306	10
20	in 5	0x306	20
21	in 6	0x306	40
22	in 7	0x306	80
23-24	???	???	??
25	+5	+5	+5

KUPER "INDUSTRIAL" CARD

The Kuper Controls "Industrial Card" is a 'half slot ISA' card,
a variation on the RTMC-48. So for OPCS, install the "RTMC48.COM"
driver.

The "H1" 40 pin connector (upper-left) is the steps/direction.
The "H2" 40 pin connector (upper-right) is the "logic" connector.
For OPCS, only the "H1" connector should be used.

On the H1 connector, pin #1 is at the lower-left of the connector
(component side facing you).

This card has 3 jumper blocks, whose "factory" settings are:

```
   JP1:  o-o o            -- sets voltage for pin #1 (OPCS: don't care)

   JP2:  o o o o o        -- sets samples-per-second(?) (default: 120/sec)
         | | |
         o o o o o

   JP3:  o o o o o o      -- sets the IRQ (default IRQ 5)
               |
         o o o o o o
```

..where '-' is a horizontal jumper, and '|' is a vertical jumper.

KUPER PORT MONITOR PROGRAM

The OPCS software comes with kuper.exe, a program that monitors the real time status of the Kuper logic port. Run 'kuper.exe'. This tool can be downloaded from http://seriss.com/opcs/ftp/

NAME

math - math expressions in OPCS

MATH EXPRESSIONS

You can usually use math expressions in place of most numeric arguments as long as the expression is ENCLOSED IN PARENTHESES, and DOES NOT CONTAIN EMBEDDED SPACES. Example:

(3+(3*sqrt(16)*12))

Math can be done on frame counter values:

(cam+3)

For a complete list of all built in math operations, execute:

(?)

The following lists some of the operations supported by the math expression parser:

```
/*** TYPICAL OPERATIONS ***/

(3+4-2*12/6)       # add, subtract, multiply, divide
(533%256)          # modulus
(2^4)              # exponentiation (powers)

/*** OPCS VALUES ***/
cam   - camera counter value
pro   - main projector counter value
pro1  - main projector counter value
pro2  - aerial projector counter value

/*** MATH FUNCTIONS ***/

sqrt(),  log(),     exp(),
sin(),   cos(),     tan(),
asin(),  acos(),    atan(),
atan2(), radians(), degrees()
hex(), pi

/*** NUMERIC EXPRESSIONS ***/
-12           # negative 12
+34           # positive 34
0x3ff         # hex representation for 1023 decimal
```

THE ONLINE CALCULATOR

To aid the operator, the above techniques can be used for printing the answer to expressions by typing them alone on a line surrounded by parentheses.

Example:

(3+(3*sqrt(16)*12))

..on a blank line (as if it were a command), the answer will be

printed to the screen:

147.000000

Camera, and both Aerial and Main Projector frame counters can all be referenced in math expressions by their command names.

For instance, to send the Aerial Projector (pro2) to the same frame as the Main Projector's (pro) current frame counter plus 5 frames:

```
pro2 >(pro+5)
```

..so if the Main Projector's frame counter is 1000, the Aerial Projector will move to frame 1005.

Or to sync both projectors to camera's current frame counter:

```
seek >(cam) >(cam) 0
     ------ ------ ------
     Aerial Main   Camera
```

..so if the camera counter is 210, this will send both aerial and main projectors to frame 210 as well.

BUGS

Needs logicals some day, like 'not()', 'xor()', 'and()', etc.

ORIGIN

Gregory Ercolano, Los Feliz California 12/15/89

```
MOV(CUSTOM)         OPCS Custom Move Commands             MOV(CUSTOM)

NAMES
    clmov   - clear all moves
    mov     - program moves using linear or ease curves
    grmov   - graph programmed moves
    usemov  - use moves created with recent mov commands
    stopmov - cancel a move in progress
    savemov - save a move to a file
    loadmov - load a move from a file

SYNOPSIS
    clmov
    mov <chans> <sfrm> <efrm> <spos> <epos> <ein> <eout>
    grmov <chans> <sfrm> <efrm>
    usemov <chans>
    stopmov
    showmov
    editmov
    savemov <filename>
    loadmov <filename>

EXAMPLE
    clmov                     # clear all moves in mov.pos
    mov e 1 50 0 10000 10 10  # create a 50 frame move on the e channel
    mov h 1 50 0 8000  0  0   # create a 50 frame linear move on h channel
    showmov                   # (OPTIONAL) shows moves created
    editmov                   # (OPTIONAL) opens move in MS-DOS editor (EDIT)
    grmov e 1 50              # (OPTIONAL) graph move on e channel, ESC quits
    usemov eh                 # start using moves in eh channels
    rep 100                   # first 50x will shoot move

DESCRIPTION
    This series of commands helps camera operators construct and modify
    move files easily.

    The commands are really custom scripts build around the external
    EASE.EXE and GR.EXE tools, and around the FEED(OPCS) command,
    to make them easier to use. You can actually do everything these
    commands do with just EASE(DOCS), GR(DOCS), and FEED(OPCS).

    MOV
    ---
    MOV creates moves using linear or ease curves for the named channels
    given a frame range and a move range, and optional ease in/out values.

        mov ef 1 50 0 10000 10 10
            -- ---- ------- -----
            |   |      |      |
            |   |      |      Ease in/out #frames
            |   |      Position start/end
            |   Frame start/end
            Channels to affect

    MOV is a script that invokes the ease.exe program to generate the
    actual moves, which can be saved to a file with SAVEMOV, and loaded
    with LOADMOV, shown with SHOWMOV, and graphed with GRMOV.
```

USEMOV

USEMOV is similar in function to the FDI/FDO command, preparing the system to begin reading values from the channels in the current position file (mov.pos). Once USEMOV is invoked, any commands that shoot the camera first moves motors to the next positions in the move file.

Once the entire move has been shot, the move file is automatically disabled. This can also be done prematurely with the STOPMOV command.

USEMOVE is a script that uses FEED(OPCS) to do the actual work.

REUSING MOVES

Once created, moves can be used more than once:

```
usemov ef rep 50
seek >120 10          # run out to next element, 10x black
usemov ef rep 50
seek >220 10          # run out to next element, 10x black
usemov ef rep 50
..
```

SAVING MOVES

Moves can also be saved for later use by copying the mov.pos file to another name:

```
mov e 1 50 0 10000 10 10      # create 50x move 0 to 10k on e chan
mov f 1 50 0 8000  0  0       # create 50x move 0 to 8k on f chan
savemov pan-to.pos            # save for later

mov e 1 50 10000 0 10 10      # create reverse move back to zero
mov f 1 50 8000  0 0  0
savemov pan-from.pos          # save for later

[..days go by, other work done..]

loadmov pan-to.pos            # bring back 'pan to' move..
usemov ef rep 50              # ..and shoot it

loadmov pan-from.pos          # bring back 'pan from' move
usemov ef rep 50              # ..and shoot it
```

SEE ALSO
FEED(OPCS) - feed new positions to motor every time camera shoots a frame
EASE(DOCS) - create ease in / out positions
GR(DOCS) - graph moves to screen

AUTHOR
Greg Ercolano, Venice California 1998

OPCS(OPCS) Optical Printer Control System OPCS(OPCS)

NAME

opcs - optical printer control system

USAGE

opcs [startup-file]

INTRO

The OPCS software is command oriented, executing commands like a unix shell. The upper half of the screen is for film counters and positions, the lower half for user entered commands.

There's at least two screen styles; 'bigcounters' which takes up half the screen, and 'small counters' which takes up much less, the latter leaving more room for the user's command history.

The heart of the software is the stepper motor driver routine which can control 4 axes per up to 3 parallel ports on the back of the computer.

STARTUP

By default 'opcs' loads the OPCSDEFS.OPC file that is in the current directory. If the optional [startup-file] is specified, that file is loaded as the OPCSDEFS.OPC file instead.

The OPCSDEFS.OPC file contains special 'OPCSDEFS' commands that set up the software, setting things like motor speeds, ports for sensing buckle/viewer, defining motor hardware settings, custom commands, etc.

To list all the OPCSDEFS.OPC commands, run '**man -k OPCSDEFS:**' and then you can focus on each one of the commands by running 'man' followed by the name of the command (e.g. '**man ramp**')

COMMAND LINE EDITING KEYS

In OPCSK200 (released August 2020), new command line editing features were added, to make it easier for operators to run and edit commands, and access a command line history to retype previously entered commands.

Edit keys are similar to modern command shells:

```
   Up Arrow -- previous line in command history      (^P)
   Dn Arrow -- next line in command history          (^N)
   Lt Arrow -- move reverse one char on current line (^B)
   Rt Arrow -- move forward one char on current line (^F)
  Backspace -- backspace and delete                  (^H)
     Delete -- delete character                      (^D)
       Home -- move to start of current line         (^A)
        End -- move to end of current line           (^E)
  Ctrl-Home -- jump to top of command history
   Ctrl-End -- jump to bottom of command history (current line)
  Ctrl-Left -- word left
 Ctrl-Right -- word right
         ^K -- clear to end of line
         ^U -- clear current line (hit again to 'undo')
         ^V -- enter next character literally
        ESC -- clear current line (hit again to 'undo')
         F3 -- re-type last command
         F4 -- re-run last command (F3 + Enter)
```

OLD MS-DOS STYLE EDITING KEYS

In the older OPCS releases (K100), only the MS-DOS editing keys were available, and were kinda funky to use, e.g.

```
          F1 -- Repeats the letters of the last command line, one by one
    Rt Arrow -- Same as F1
         Ins -- Enables you to insert characters into the line
         Del -- Deletes a character from the line
          F2 -- Copies all characters from the last command buffer
                up to, but not including, the next character you type
          F3 -- Copies all remaining characters from the preceding command line
          F4 -- Usually re-programmed in AUTOEXEC.BAT to re-run last command
          F5 -- Moves current line to buffer but doesn't execute it
```

ONLINE DOCUMENTATION

Use **man -k opcs** to get a list of all OPCS related documentation, or **man -k opcs:** for just the OPCS commands themselves.

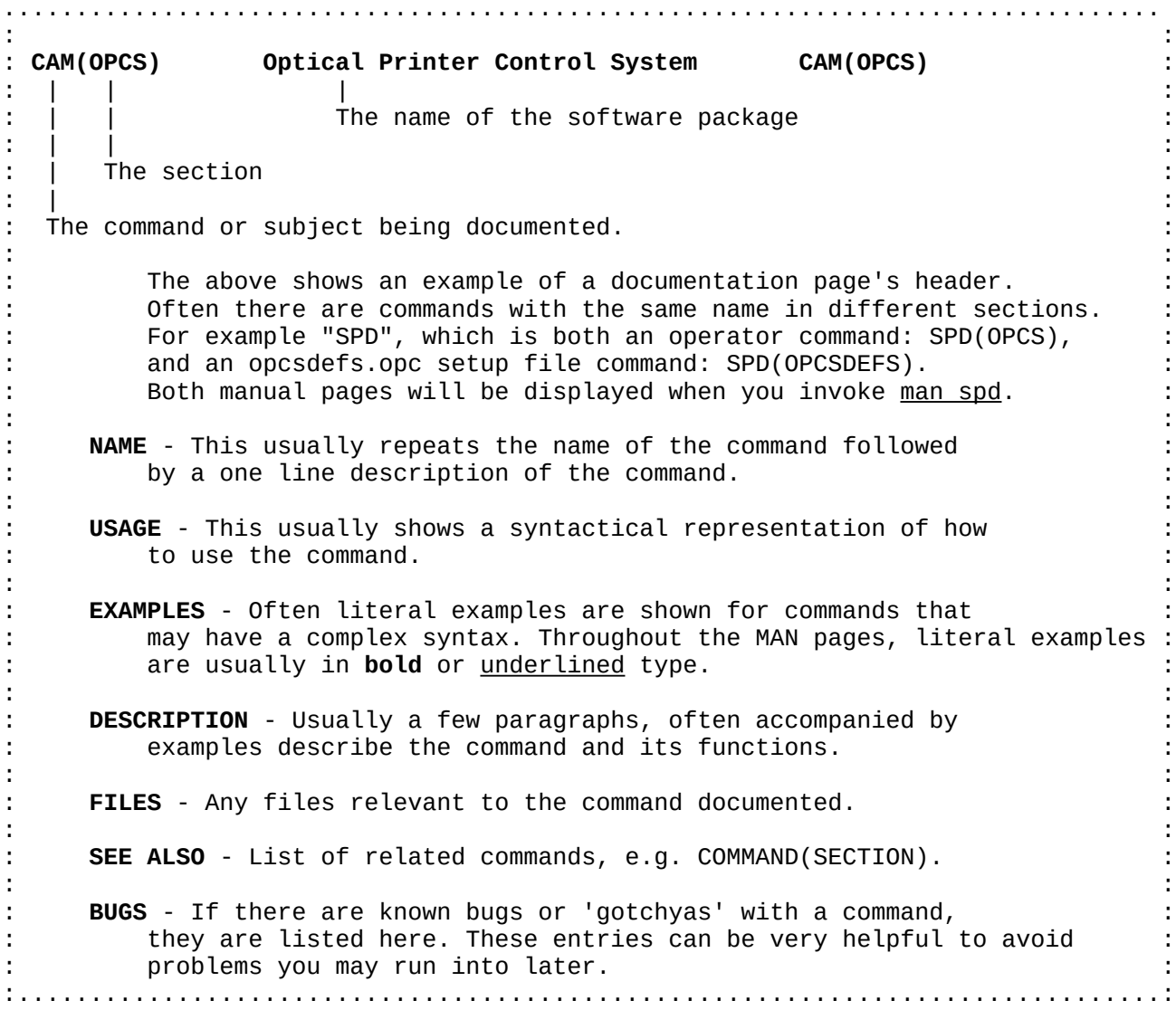
The same goes for `**man -k opcsdefs:**` which will list just OPCSDEFS commands that are available.

From the print out of these commands, you can determine which commands to pull up a full man page on. Just type 'man' followed by the name of the command you want to know more about:

```
    man cam       # find out more about the CAM(OPCS) command
```

Using this technique, you can guide yourself through all the OPCS commands, and how they interact.

All the OPCS documentation pages follow a similar format. EXAMPLE:



Documentation pages are viewed with the MAN command, and texts may be customized by the end users if they wish.

TIPS

For additional help on how to formulate commands, and shortcuts for typing commands, refer to SYNTAX(DOCS) (i.e. '**man syntax**'), which describes the online calculator, command stacking, and other shortcuts.

ERRORS

Most errors are self explanatory, but some need extra explanation, covered below:

FADER NOT AT ### DEGREES FOR FDI/FDO/DXI/DXO

When trying to do a fade/dx in, the shutter must first be completely closed, and when trying to do a fade/dx out, the shutter must be completely open.

FILM BUCKLE or VIEWER OPEN
HIT ENTER to CONTINUE or ALLSTOP to ABORT:

While trying to run the camera, the film buckle switch has been tripped or the viewer is open.

RECURSION ERROR

The RUN command prevents you from running a script that is already running, to prevent infinite recursion.

A run script calling itself, or a child script that calls a parent is considered 'recursion'. Here a script calls itself:

```
# fred.run
cam 1 pro 1
run fred.run  <-- FAILS HERE
```

In this example, a child script calls one of its parents:

```
# test1.run
fdi 10 rep 10
run test2.run

# test2.run
fdo 10 rep 10
run test1.run  <-- FAILS HERE
                   The way to get the desired result
                   is to remove this line, and start
                   the scripts by executing e.g.:
                   DO 12 RUN TEST1.RUN
```

NESTED TOO DEEP

Too many run script levels. When a script calls another script, that is '2 levels' of nesting. Up to 10 levels of nesting are allowed before this error occurs.

STOPPED AT LINE (#) OF (#)>(filename)

An error occurred in a run script, and this message indicates the line number, nesting level, and the name of the script where the initial error occurred. One message per nesting level is printed, with the FIRST MESSAGE being the script containing the error.

INVALID REPEAT COUNT

In a DO command, the value specified was negative, or not a number.

SPEED OUT OF RANGE OR INVALID

In a SPD command, the number specified was too low (0 or below) or too high. Normally, the software cannot run the motors slower than a 10 second exposure speed, but depends on the motor ramping and acceleration values. Have you recently changed them?

UNKNOWN OPCSDEFS COMMAND
A command in an OPCSDEFS file was invalid.

INVALID FEET/FRAMES
A specified feet/frames value was invalid. Usually, the frames value exceeded the number of frames in a foot.

MISSING ARGUMENT AFTER 'command'
The software expected an argument where one wasn't supplied.

SYNC FAULT
The software was not able to keep up with the hardware. Get a faster computer, seriously.

This is a fatal error where the motors probably stalled because the software couldn't feed velocities fast enough to the motors, i.e., if the cpu is too slow. Or some other hardware/driver is using up the cpu by generating interrupts.

This error can also occur when debugging is enabled; debugging messages can sometimes slow the software down enough to where it can't update the motors quickly enough.

SPD: SPEED TOO SMALL (IGNORED)
The resulting speed set by the SPD(OPCS) command would have been a value too small.

FILES

\OPCS\BIN\OPCS.EXE	- the executable program
\OPCS\BIN\MAN.EXE	- online documentation program
\OPCS\MAN*	- online documentation pages
OPCSDEFS.OPC	- the 'start up' definition file
***.OPC**	- other opcs definition files
***.RUN**	- RUN(OPCS) scripts
***.LOG**	- LOG(OPCS) files
***.VRP**	- 'VELREP(OPCS)' files

ENVIRONMENT VARIABLES

OPCSDEFS=<hex>	- pointer in memory to OpcsDefs structure
OPCSLOOP=#	- DO loop iteration (0 if none)
OPCS_NOMOTOR_FRAME_DELAY=#	- Delay for simulated frames when ‘motors off’

SEE ALSO

QUICKREF	- OPCS camera operator quick reference/tutorial
SYNTAX	- Online calculator and OPCS math expression syntax
OPCSSETUP	- OPCS setup/installation procedures
OPCSHARDWARE	- OPCS hardware specifics (wiring, etc)
VERSIONS	- OPCS version information (a list of all revs)
man -k OPCS:	- list all OPCS operator commands
man -k OPCSDEFS:	- list all OPCSDEFS file commands

ORIGIN
Gregory Ercolano, Los Feliz California 01/18/91

OPCSDEFS(DOCS) Optical Printer Control System OPCSDEFS(DOCS)

NAME

opcsdefs - OPCS definition file format description

DESCRIPTION

When the opcs program is started, the 'OPCSDEFS.OPC' is loaded.

This file contains commands that sets the motor and ramp speeds, channel names, custom commands, interpolation curves, etc.

After startup, the LDEFS(OPCS) command can be used to load different files containing other OPCSDEFS commands, allowing users to switch to different configurations, such as switching from IMAX to 35MM shooting, or changing around a filter wheel configuration, loading different follow focus files, fader curves, etc.

The commands in OPCSDEFS files are different from the OPCS(OPCS) commands, and even when names are similar between them (e.g. 'reset'), their usage and context may be quite different, such as the case with SPD(OPCS) vs. SPD(OPCSDEFS).

FILE FORMAT

Custom files with OPCSDEFS commands should use the extension ".def", so as to be different from files with OPCS commands which use ".run".

Lines starting with '#' and blank lines are ignored.
Comments can also appear after commands, e.g.

```
ramp a 10 150 15 200   # 'A' channel ramps
```

Leading/trailing white space in generally ignored, so you can indent commands for formatting.

Multiple commands can be stacked on a line if it serves readability, e.g.

```
# BUCKLE SENSING PORTS
buckle a 0000 00 00     buckle e 0000 00 00     buckle i 0000 00 00
buckle b 0000 00 00     buckle f 0000 00 00     buckle j 0000 00 00
buckle c 0000 00 00     buckle g 0000 00 00     buckle k 0000 00 00
buckle d 0000 00 00     buckle h 0000 00 00     buckle l 0000 00 00
```

..but /generally/ there should be only one command per line.

Commands that start with "!" will be execute as DOS commands.

OPCS commands can be run from within an OPCSDEFS file using OPCSCMD(OPCSDEFS), e.g.

```
opcscmd cam 12         # run the camera 12 frames
opcscmd go d -1000     # move the fader -1000 pulses
```

OPCSDEFS files can load other OPCSDEFS files with e.g.

```
opcscmd ldefs otherfile.defs
```

DOCUMENTATION

Use **man -k OPCSDEFS:** for a full list of the OPCSDEFS commands. With this list, you should be able to zero in on specific commands using **man [command]**.

TRICKS WITH DEFS FILES

People familiar with the IBM's operating system will be familiar with these capabilities...

First, note that in K2.00 (and up), '**ldefs -c**' can be used to run OPCSDEFS commands inside OPCS, e.g.:

```
ldefs -c bigcounters on   # big counters
```

Which makes many of the below techniques unnecessary extra work. However, in the older releases (K1.xx) these are unavailable, so the below techniques must be used.

As with all DEFS file commands, you can execute motor definition commands from within the OPCS software by creating a small file, and the loading commands from it via LDEFS(OPCS)... In the following example, we switch back and forth between large and small counters:

```
! echo bigcounters on  > tmpfile ! ldefs tmpfile   # big counters
! echo bigcounters off > tmpfile ! ldefs tmpfile   # small counters
```

This 'trick' can be used with any OPCSDEFS commands, and uses the operating system's ECHO command and 'reroute output' symbol (>) to create the file FOO, which is then loaded as a file with the LDEFS command. This technique CAN be used within a script or when entering commands manually.

You can create multiline files from within a script as shown in this example using MSDOS's > and >> (append) symbols:

```
! echo flog 2.0        > tmpfile
! echo logcounters yes >> tmpfile
ldefs  tmpfile
```

This technique can be programmed into run scripts, so defs file information can be changed on the fly.

Here is another way to enter DEFS commands directly to the LDEFS command from within the OPCS software:

```
ldefs con           # Load the special MSDOS file CON...
logcounters no      # which is really the keyboard (console)
ppr a 400           # reading these commands from keyboard
^Z                  # CTRL-Z and RETURN ends this mode..
cam 12              # ..back to OPCS commands
```

The '**ldefs con**' technique works well for interactive typing, but cannot be programmed into a script, since it always reads from the keyboard. Use the 'echo' technique listed in the previous example for programming DEFS commands into a running script.

These techniques are actually standard ways of using the DOS operating system, and are not particular to just the OPCS software. They can be used by any program running under MSDOS that properly supports the operating system.

Users not familiar with these techniques should learn them only if they think they might need them. At very least, operators should be aware of these capabilities.

FILES

\USR\BIN\OPCS.EXE	- the OPCS system software executable
OPCSDEFS.OPC	- the 'start up' definition file
***.DEF**	- other opcs definition files
***.RUN**	- run scripts
\USR\CATMAN\OPCS*	- online documentation pages

SEE ALSO

OPCS(DOCS)	- the opcs system in general overview
OPCSHARDWARE(DOCS)	- hardware specifics (wiring, etc)
OPCSIFACE(DOCS)	- OPCS interface boards (A800, PIO-100, SD-800..)

ORIGIN

Gregory Ercolano, Los Feliz California 11/29/89

OPCSETUP(DOCS) Optical Printer Control System OPCSETUP(DOCS)

NAME

opcsetup - opcs setup notes

VERSION OPCSK2.00

GENERAL

This text describes how to set up the OPCS software from scratch, especially for a NEW installation. To greatly simplify this text, it is assumed you know certain DOS techniques and terminology that is available from your DOS manual, such as copying files to and from floppies, what subdirectories are, how to create them, execution PATHs, etc.

It is assumed you have:

- A KUPER or A800 stepper control card plugged in
- An IBM PC with ISA or EISA slots
- 512K or more of system memory.
- Running MS-DOS 6.xx or Win95 or Win98 to boot in DOS mode
- A hard disk (needed for the online manual pages)
- AUTOEXEC.BAT and CONFIG.SYS (see below)

CONFIG.SYS

Your CONFIG.SYS should at minimum have these (or similar) settings:

```
DEVICE=C:\WINDOWS\HIMEM.SYS
DEVICE=C:\WINDOWS\COMMAND\ANSI.SYS
DEVICE=C:\OPCS\BIN\OPCSBOLD.SYS
FILES=10
BUFFERS=20
```

> HIMEM.SYS pushes DOS into high memory (>640k) allowing apps like OPCS to have more ram to run in the lower 640K. Use this if your machine has extended memory installed (most computers do).

> ANSI.SYS provides text colors/highlighting which OPCS uses

> OPCSBOLD.SYS (OPCSK200 and up) provides the special character set needed for '**bigcounters nixie**' in OPCS version K2.xx and up. (Leave this line out if your OPCS version is K1.xx)

> The FILES and BUFFERS lines help speed up disk access by allowing ram caching to be used. Optional but generally recommended.

Note HIMEM.SYS and ANSI.SYS come with the operating system, so they should be present already. OPCSBOLD.SYS comes with the newer OPCS versions (K2.xx and up), so only include that line if your OPCS version has that file.

AUTOEXEC.BAT

Your C:\AUTOEXEC.BAT should add the OPCS 'BIN' directory to the PATH, and start the correct driver for the stepper card you're using. For instance, if you have the OPCS K2.XX version and an A800 card:

SET PATH=\OPCS\BIN;%PATH% -- add the opcs 'bin' to the PATH
SET MANPATH=\OPCS\MAN\MAP -- sets up the OPCS 'man' pages
A800DRV -- start the A800 driver with default settings
CD \OPCS\WORKA800 -- leave DOS in the A800's WORK directory

These are the different drivers OPCS supports for stepper motor cards; use these in place of the A800DRV command above:

A800DRV.COM - for the OPCS a800 card
RTMC48.COM - for the Kuper RTMC48 or Kuper Industrial card
MDRIVE.COM - for the Kuper RTMC16 card

If started with no command line flags, the drivers assume the cards are using the default jumper settings (usually baseaddr=300, IRQ=5).

OPCS K2.10 (and higher)

In K2.10 and up, the A800DRV, RTMC48, and MDRIVE motor drivers support options to set the IRQ and/or BaseAddr to something other than the defaults. Run the driver with the -help flag to view the options:

a800drv -help
rtmc48 -help
mdrive -help

NOTE: If your driver doesn't show a help screen, that driver ONLY supports the default jumper settings.

So if your A800 card has the BaseAddr jumpers set to 340 (instead of default 300) then you must specify the -b flag, e.g.

a800drv -b340

|
Sets BaseAddr to 340

Similarly if you have the IRQ jumper on the A800 set to IRQ6 (instead of default IRQ5) then you must specify the -i flag, e.g.

a800drv -i6

|
Sets the IRQ to 6

On OPCS K2.10 and up, the RTMC48 and MDRIVE drivers have similar option flags.

C:\MSDOS.SYS

On Windows 95/98 machines, you should disable Windows from starting so the machine boots directly to DOS. To do this, run these commands:

1) **ATTRIB -R -H -S C:\MSDOS.SYS**
   ..which un-hides the file so you can edit it.

2) **EDIT C:\MSDOS.SYS**

3) Under [OPTIONS], adjust so you have these settings:
   BootGUI=0
   Logo=0
   BootDelay=0
   Where:
   BootGUI=0 -- disables Windows from starting automatically
   Logo=0 -- disables the Windows splash screen
   BootDelay=0 -- prevents any delay during booting:

4) Save changes

5) **ATTRIB +R +H +S C:\MSDOS.SYS**
   ..which re-hides the system file so it is used on boot.

VERIFICATION

To verify that the software is installed properly, reboot the system (so changes to the AUTOEXEC.BAT and CONFIG.SYS file take effect), and watch for any errors, and fix them.

Then run OPCS. The software should start, printing the copyright banner, and indicating the OPCSDEFS file loaded properly. Look for error messages and fix them.

When properly operating, the large counters should show up on the screen, and you should get a triangular arrow prompt.

At any time, you can type 'q' or 'qq' to quit the software to return to DOS.

If you see any error messages, the software either cannot access the OPCSDEFS.OPC file which should be in your current directory, or there are errors in the file. Use the text editor to fix or otherwise customize the OPCSDEFS.OPC file.

Execute 'man cam' from within the software to verify that the online manual has been installed properly.

If so, documentation on the 'CAM' command should come up with a MORE prompt at the bottom of each page. Hit 'SPACEBAR' to advance a page, 'B' to go back a page, or 'Q' to quit. Hitting RETURN will advance single lines. In newer versions, Up and Down arrow and/or 'J' and 'K' will move up/down one line.

If you see any errors while trying to run the MAN command:

'Bad command or filename'
Either the 'MAN.EXE' command or 'MORE.EXE' is not in the machine's execution path. Make sure these files are in your executable \BIN directory, and the directory is properly specified in DOS's execution PATH. Use the DOS 'set' command to check.

'man: could not open map'
The MANPATH environment variable is not set or points to a directory that doesn't exist or doesn't contain a MAP file.

NOTHING HAPPENS
MAN may be using DOS's inferior MORE program to view the manual pages. Type ^C or ^BREAK to break out of DOS's MORE program, and make sure the \BIN directory is in the execution path before the DOS directory is. Example:

If your execution path is setup in your AUTOEXEC.BAT file to check the DOS directory before checking the BIN directory, such as:

PATH=C:\DOS;C:\OPCS\BIN

Then change the order so the opcs BIN directory is first, e.g.

PATH=C:\OPCS\BIN;C:\DOS

TUNING THE SOFTWARE FOR NEW HARDWARE

It is probable that the software should now immediately be able to make motors spin round and round, assuming the hardware is connected correctly. Maybe not smoothly, or making complete revolutions, but that comes next.

You must now spend some time jumping in and out of the software, tuning your OPCSDEFS.OPC file to suit your hardware. Keep in mind that each time you make a change to the OPCSDEFS.OPC file, you must either rerun the OPCS software, or execute 'ldefs opcsdefs.opc' to force the software to recognize the changes.

The following checklist should help you correct most problems:

- If your motors stall when trying to run them at speeds that they SHOULD turn at, you may want to tune the RAMP(OPCSDEFS) and SPD(OPCSDEFS) values in your OPCSDEFS.OPC file

  If you think the motors may just be running too fast, modify the SPD(OPCSDEFS) commands in your OPCSDEFS.OPC file to run the motor slower. See man pages on this command for details.

- If frame-oriented motors are not making complete revolutions, alter the PPR(OPCSDEFS) command to change the number of pulses in a revolution for your motor. Most shutters need 2000 pulses to revolve one full turn when using micro stepper drives. For those of you with VISTAVISION shutters, 4000 might be more suitable.

- If a motor runs reverse when told to run forward, and vice versa, change the DIRXOR(OPCSDEFS) value for that motor. (see man pages on DIRXOR). This command allows you to invert the direction of a motor without modifying the hardware.

- If your fader does not fully open, fully close, or does not do linear dissolves properly, see the INTERP(OPCSDEFS) documentation for setting interpolation positions for every 10 degrees on the fader.

- If you don't like the initial speed the software comes up with for the camera or the default running speed for the projector, see the SPD(OPCSDEFS) documentation (which will come after the SPD(OPCS) docs).

- If the fader appears to suffer from hardware slop (ie. the sequence 'cls shu 150' and 'opn shu 150' do not send the physical shutter to the exact same position, even though the motor does not appear to stall) this is due to mechanical hysteresis in the shutter mechanism, and can usually be alleviated ENTIRELY by use of the SLOP(OPCSDEFS) command, even in systems where slop of 5 to 10 degree deviations (typical of most old printers) is found. SLOP is a cool command, and can make really sloppy hardware work very accurately.

Once you have tuned the system, and wish to start learning the commands, type ? in the software to get a list of all available commands, and read the online MAN pages for each command that interests you.

SEE ALSO

OPCSDEFS(DOCS) - OPCS configuration file
OPCSHARD(DOCS) - hardware specifics (wiring, etc)
OPCSIFAC(DOCS) - OPCS interface boards (A800, PIO-100, SD-800..)

ORIGIN

Gregory Ercolano, Los Feliz California 11/29/89

```
OPCSHARD(DOCS)      Optical Printer Control System       OPCSHARD(DOCS)

NAME
    opcshard - notes on rigging the OPCS hardware

OPCSK100 and OPCSK200 Software
------------------------------
    The OPCS system uses either the A800 or one of the Kuper Controls
    cards (RTMC16, RTMC48, Kuper Industrial) to generate pulse streams
    to run the steps/direction inputs on the microstepper motor drives.

           PULSE GENERATOR ISA CARDS
           -------------------------
           RTMC16 -- Kuper 16 axis 'full size' card using discrete
           RTMC48 -- Kuper 48 axis 'full size' card using FPGAs
           Kuper Industrial -- Kuper 16 axis 'half size' card using FPGAs
           A800             -- OPCS 8 axis 'half size' card using PICs

    A variety of stepper motor drives can be used with the above cards:

           STEPPER MOTOR DRIVES
           --------------------
           Centent (CNO-142, CNO-143, CNO-162, CNO-165)
           Gecko (201 and 201X)
           Leadshine (DM-542)
           Sanyo (FMD2740C)

    Although not required, the following OPCS interface boards can be
    used to simplify wiring to the motors and digital sensors:

           ANCILLARY CARDS
           ---------------
           PIO-100  - Parallel I/O interface board (see 'man pio-100')
           SD-800   - Stepper Distribution board (see 'man sd-800')

    The PIO-100 board is connected to the computer's parallel port,
    and breaks the signals out to individual RJ-45 ports, allowing
    RJ-45 patch cables to run out to each digital sensor, e.g.
    home sensors, buckle/viewer switches, deenergize on the step drives,
    tension motor controls, etc. This simplifies wiring, and allows
    sensors to be easily reassigned.

    The SD-800 board is connected to the computer's step pulse generator
    (e.g. RTMC16, RTMC48, Kuper Industrial, OPCS 'A800' board..), and
    fans out the step/direction signals for each channel to individual
    RJ-45 ports, allowing RJ-45 patch cables to run out to each channel's
    stepper drive. This simplifies wiring, and allows motor channels to
    be easily reassigned.
```

OPCS SOFTWARE REQUIREMENTS/LIMITATIONS

In order to run the optical printer effectively, the first 4 channels (a,b,c,d) are pre-assigned in the software for specific purposes:

KUPER CHANNEL	OPCS CHANNEL	DEVICE EXPECTED TO DRIVE
0	A	Aerial Projector
1	B	Main Projector
2	C	Camera
3	D	Fader

Other channels (E, F, etc) have no requirements, and can be assigned to any purposes, such as zoom, focus, east/west and north/south pan, filter wheels, capping shutters, etc.

The OPCS software was originally designed to control a maximum of 12 motors. Even though the Kuper card can control up to 16 axes, the OPCS software can only drive a maximum of 12.

OPCSk1.00 does NOT use the Kuper card's encoder feedback. Under normal use, motors should never stall or lose position unless there is some hardware problem (e.g. stuck gears, bad connections, frozen equipment, bad pulley and/or gearing ratios), or the ramping values were not set properly (see RAMP(OPCSDEFS), SPD(OPCSDEFS), etc).

The software provides for inverting the direction of a motor if it runs in the wrong direction. See DIRXOR(OPCSDEFS) to correct this. Normally, if the DIRXOR bit for a motor is 0, the motor will turn clockwise when told to run 'forward' from the software. By changing the DIRXOR bit to a 1, telling the motor to run 'forward' will make it run counter-clockwise.

The OPCS software's definition file (OPCSDEFS.OPC file) can program any of the IBM's hardware ports to control/monitor the following functions. Usually the parallel port is used for this, though 3rd party digital I/O boards (such as the 8255 based I/O boards) can be used as well:

Function	DEFS command
Film buckle	buckle
Viewer Open	viewer
Deenergize (unlock motors)	deenergize
Allstop	allstop
Motor Direction Inversion	dirxor
Home Sense	(see HOME SENSING below)
Set a bit on a port	setbit
Clear a bit on a port	clrbit
Invert a bit on a port	xorbit
Tension motors	tension

The software currently requires at LEAST a 25Mhz machine, or faster to properly update the motors. The software will display the error:

FATAL ERROR: Sync Fault (probably lost positions)

...accompanied by some disagnostics data if it finds the CPU cannot keep up with the motors.

The software relies on the Kuper card's timebase to compute accurate camera exposures. Therefore, the software will have exposures consistent from machine to machine, regardless of the CPU's speed.

HOME SENSORS

"Home sensors" or "optical sensors" allow the software to find each channel's "zero position" automatically from software.

> Camera shutter needs to home in the "closed" position
> Projector shutters needs to home in the "seated" position
> Fader should be homed in the CLOSE position
> Pan and zoom should home in the 1:1 center position
> Filter wheels should be home in the full open (no filter) position
> Capping shutters should home in the open position

Home sensing is handled by the external 'home' program. It has its own setup file, HOMEDEFS.HOM that defines which computer port bits are associated with which home sensor, and which home sensor with which motor channel. This file also defines the motor homing routines; a simple 'scripting langauge' the defines how each motor channel should find home.

Typically when OPCS first starts up, all motors are homed automatically via commands near the bottom of the OPCSDEFS.OPC file that sends each channel home, and zeroes the software counters. e.g.

```
! home a b c d      # home the a/b/c/d channels
reset abcd 0        # reset the software counters to zero
```

See 'man home' and the HOMEDEFS.HOM file for examples of how the home program can be customized.

Users can define their own OPCS commands to home the motors, using either RUNCMD(OPCSDEFS) or DOSCMD(OPCSDEFS) to run external programs such as the 'home' program.

Home sensors usually manage either rotational or linear motion oriented channels:

> Rotational sensors usually sense a slot in a disk to find home

> Linear sensors usually sense the edge of a bar of metal running half the length of linear motion to find home. See "PAN CHANNELS" (below) for more info.

AERIAL & MAIN PROJECTOR

The projector(s) usually have an M0-63 type motor to run the film movements, and Bodine tension motors to keep feed and takeup tension on the film. Home sensors on the movement ensure when the software starts up, the film movement is in the proper, fully seated position.

There's usually one pair of tension motors for the regular film path, and a separate pair of tension motors for a secondary film path, such as when bipacking in the projector.

The home position for projectors should be in the fully seated position, so that looking through the camera viewer will see a fully

seated projector image.

Typically motors are geared 1:1 to the film movements, which is to say one turn of the motor moves the film movement one full cycle, which is usually a full frame advance (for 35mm film), or sometimes a fraction of a frame for the larger format film (e.g. IMAX, Vistavision).

For larger format films that involve several pulldown cycles of the film movement to expose a single frame, a capping shutter is utilized to expose only on one of the cycles that is the film exposure, and caps out the intermediate pulldown motions involved in advancing to the next frame in fractional steps.

Usually it's best to run the projectors at their maximum safe speed, which is usually around 20 feet per minute, which is usually around 0.25 to 0.18 exposure speed (around 5 frames per second).

CAMERA

The camera film movememnt ususally uses an M0-63 type motor to run the film movements, and Bodine tension motors to keep feed and takeup tension on the film, a separate pair of tension motors for raw stock and optional bipack.

The home position for the camera should be in the fully CLOSED position, so that the camera is NOT exposing film when sitting idle.

FADER

The fader on optical printers is often difficult to configure for stepper motor control, especially cameras that need linear movement to rotate the shutter blade.

Also many faders have a built in logarithmic movement that has to be counteracted for proper computer controlled dissolves. It's therefore often the case an interpolation curve is needed to undo the logarithmic motion. This can be done with the INTERP(OPCSDEFS) command. (See 'man interp' for more info)

Faders are typically 170 degrees, which is the number of degrees the fader shutter's opening is. This is 10 degrees less than 180, which allows 5 degrees of overlap on either end of the fader with the camera's shutter, to ensure no light leaks around the fader shutter when it's fully closed.

The home sensor for the fader is usually positioned such that the fader homes in the closed position.

The fader is often used as a cap, to wind off black frames, and then moved to full open to do normal shooting.

There are three camera operator commands that directly move the fader shutter:

```
opn    -- open the fader
cls    -- close the fader
shu 50 -- move the fader to specific positions in degrees
```

Normally the 'interp d ..' OPCSDEFS command is configured to

convert degrees into actual step positions. For a linear shutter, this would be a simple command such as:

```
    interp d - 0 170 2 0 12000
           | | |  |  | |   |
           | | |  |  | |   End step position (open)
           | | |  |  | Start step position (closed)
           | | |  |  Number of step positions in the interpolation
           | | |  End position in degrees (open)
           | | Start position in degrees (closed)
           | (No slaving channel)
           The fader channel
```

Finding Fader 10 Degree Positions

For logarithmic faders, setup involves removing the camera face plate to expose the actual fader and camera shutters so that one can mark 10 degree positions on the camera body.

1) Remove the camera's face plate, exposing the camera/fader shutters

2) Make sure there is no interpolation already configured for the fader channel by running:

```
    ldefs -c interp d - 0 0 0
```

3) Home the camera and fader, and reset the fader's software counter to zero:

```
    ! home c d
    reset d 0
```

   The fader should now be fully closed, and the camera shutter should be in the closed position, where the center of the camera's shutter is covering the light path to the film.

4) Jog the fader using:

```
    jog d
```

   ..until the fader is in the full open position. Make note of the fader's step counter, as that will be the 'full open' position. For the purposes of an example, let's say 'full open' is 54100.

   Use the ESC or 'q' key to break out of jog mode.

5) Send the fader to the closed position using:

```
     go d >0
```

   Using a protractor, mark on the front of the camera the degree positions, starting with "0" for the leading edge of the fader in the fully closed position, then using the protractor, mark every 10 degrees with a fine line on the front of the camera body.

   Label each mark in degrees: 0, 10, 20, etc. until you reach the full open position which should be 170.

   There should be 18 marks total, including zero.

6) Make a table on a piece of paper for all the 0 to 170 degree positions. You'll fill out this table in the next steps:

DEGREES	POSITION	
0	0	<-- usually always zero
10		
20		
30		
40		
50		
60		
70		
80		
90		
100		
110		
120		
130		
140		
150		
160		
170	54100	<-- this value from step #4

7) Send the fader to the closed position using:

go d >0

8) Now using 'jog d', move the fader to find each 10 degree position, moving always in the same direction (to prevent slop).

Write down the step count shown for each 10 degree position.

9) Repeat step 8 for every 10 degree mark until you reach 170.

   You should now have a table of numbers that can be plugged
   into an interp command for testing:

DEGREES	POSITION
0	0
10	8200
20	11600
30	14100
40	16800
50	19100
60	21400
70	23600
80	25600
90	27850
100	30350
110	32600
120	35100
130	37500
140	40100
150	43600
160	47850
170	54100

10) Using the table you've prepared in step #9, create an interp
    command for the 'd' channel by editing the OPCSDEFS.OPC file,
    and add the command near the bottom of the file in the 'FADER
    AND FOCUS' section. For the above example that would be:

```
interp d - 0 170 18
           0 8200 11600 14100 16800 19100 21400
           23600 25600 27850 30350 32600 35100 37500
           40100 43600 47850 54100
```

    Refer to the manual page for the INTERP(OPCSDEFS) command
    ('man interp')

11) Now reload the OPCSDEFS.OPC file so that the new interp command
    is configured by running:

```
ldefs opcsdefs.opc
```

12) Home the 'd' channel and reset the counters using:

```
! home d
reset d 0
```

    NOTE: You may want this to be automatic on startup by adding
          the following commands to the OPCSDEFS.OPC file:

```
! home d
opcscmd reset d 0
```

13) Check the degree positions work by using the SHU(OPCS) command to go to every 10 degree mark:

```
shu 10
shu 20
shu 30
..
shu 170
```

Make sure the fader's leading edge reaches each of the 10 degree marks accurately.

If slop in the fader is a problem, configure slop correction for the 'd' channel using the SLOP(OPCSDEFS) command. This sets the number of steps for "slop correction". Refer to the man page for more info.

14) Once verified, using 'opn' and 'cls' should also reach the 170 and 0 marks respectively.

That's it.

You should now be able to do lap dissolve tests to check for problems. When properly configured, cross dissolves between two gray fields should not get brighter or darker during the dissolve.

If you do see pulsation, try to track down the problem by more carefully monitoring the 10 degree marks on the front of the camera.

FILTER WHEELS

Filter wheels are large disks that hold a variety of filters in front of the film path, allowing for doing quick wedges of a set group of filters on different scenes, or allowing computer controlled filter changes during shoots, where filters are preloaded into the wheel, and OPCS scripts are devised to move to different filters for different scenes automatically.

Filter wheels should have a single position with NO FILTER, and that position should be the 'home' position, so that the camera operator can look through the viewer and see the projector footage without a filter in the way.

Filter wheels come in different sizes, from 4 position on up to many 10s of filter positions.

For printers doing YCM printing, a 4 position filter wheel is essential to handle the three yellow/cyan/magenta filters needed to either combine or split out a color image into 3 separate black&white separations.

Special YCM shooting files can be used to run the shutters in such a way that the film can be moving as fast as possible for the 3 separate YCM exposures. See 'man velrep' for creating custom motor velocity files that can run the camera/projector/ filter wheel/capping shutter all in sync for the fastest YCM shooting possible.

CAPPING SHUTTERS

Capping shutters are often used when working with film movements that involve several seat/unseat operations per frame, such as YCM footage, or large format film such as Vista or IMAX where the film movements need to cycle several times to move one frame.

Capping shutters should be *rotational* shutters: a 1/2 disk such that 1/2 the disk blocks the light (acting as a cap), and the other 1/2 exposes light.

Non-rotational shutters should be avoided, such as:

CAPPING SHUTTERS TO AVOID

Solenoid driven caps
Aperture oriented caps (blade shutters, e.g. Uniblitz)

These wear out prematurely, as motion picture work involves tens of thousands of exposures per day, which is easy to wear out devices with MTBF rating of only a million cycles. (If the average use is 10,000 frames per day, a million cycle limited shutter would wear out in 100 days)

It is advised capping shutters home in the open position, so that the camera operator can always look through the viewer to see the image in the projectors when the system is idle.

PAN CHANNELS

Pan channels for the lenses are typically small motors (M0-61 or M0-62), that do simple linear motions.

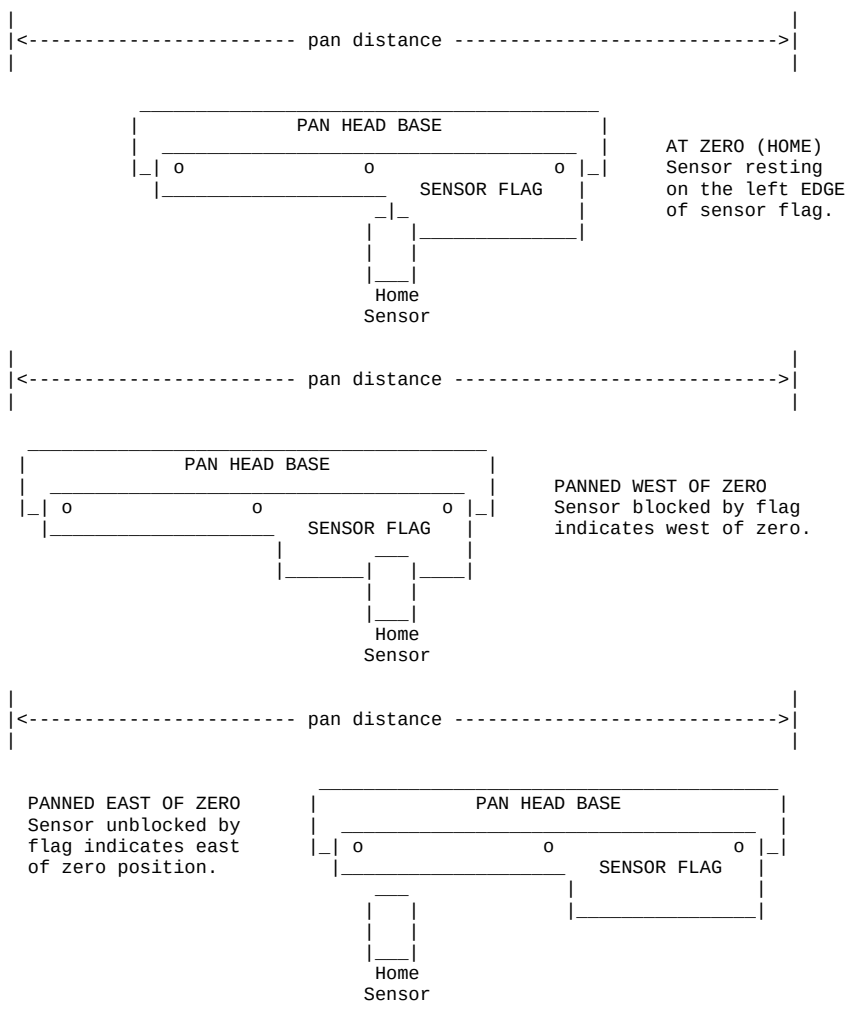
A stationary home sensor for the pan channel(s) should be configured such that the sensor is mounted to the stationary base of the printer if possible (avoiding cable movement which can fatigue the cable and connection to the sensor) and a bar of thin metal mounted to the moving pan head that acts as the "sensor flag", blocking the sensor whenever the pan head is to one side of the center (zero) position.

<IMG SRC="http://seriss.com/opcs/gifs/linear-anim.gif">

For example, an east-west pan head where "home" is the center position, and the pan head can be moved east or west of that position.

If the "sensor flag" blocks the sensor whenever the head is positioned west of center, this makes it easy to know which direction to move the motor to find home during motor homing:

> If the sensor is blocked, we're WEST of home and need to move east
> If the sensor is unblocked, we're EAST of home and need to move west



"Home" would be finding the "edge" of the bar, where it transitions from one state to another.

To prevent hardware slop from causing a variance in the home position, it's always best to always find home from moving in the SAME DIRECTION. Example: we decide to always find home moving EAST. To home the channel:

> If we're WEST of home, move EAST until we see a transition and then stop. This is the home position.

> If we're EAST of home:
>
> 1) Move WEST until we see a transition and stop.
> 2) Move a little more (*) WEST until we're past home.
> 3) Now run EAST until we see a transition and stop

(*) In step #2 when EAST of home, the extra west movement should be a little more than the amount of slop known for this hardware. So if the slop amount is approx. 800 steps, use 1200 steps for that extra movement.

Note that the most common situation when homing a motor will be when the motor is already at the home position (resting on the edge), it might be wise to assume this case, and first move the pan head off the edge of the sensor by the slop offset (described above), so that we can then approach the home sensor in the proper direction with any slop removed.

ZOOM/FOCUS CHANNELS

The configuration of the home sensor for zoom/focus should be similar to the PAN CHANNELS (described above).

For zoom there are two special considerations for setup:

- Follow focus
- Exposure compensation

FOLLOW FOCUS

In OPCS, follow focus is implemented by using interpolation between empirically determined focus points. Which is to say, during setup, you pick a fixed number of steps between interpolation points, and move the zoom channel to each position, and find focus, recording the focus positions for each fixed zoom position.

Example: Let's say the zoom's entire travel is from step position -40000 to +20000, and 0 (zero) is the 1:1 home position. This means the total zoom distance is 60000 steps.

And we decide finding focus positions for every 10000 steps works best. So that means there will be 7 focus positions (including zero) to find. (60000 / 10000 = 6), then plus one for the zero position.

Let's say our channel assignments are 'e' for zoom (camera lens), and 'f' for focus (camera base).

Make a small table showing all the zoom positions from -40000 to 20000 in increments of 10000, and a separate column for the focus positions we're going to find:

ZOOM(E)	FOCUS(F)	
-40000		
-30000		
-20000		
-10000		
0	0	<-- we know this is zero already
10000		
20000		

We now go through the repeating process of finding the focus positions for each of the zoom positions:

Finding Focus Positions
---

1) Start by making sure there are no interpolations already configured for the zoom and focus channels by disabling any existing interpolations by running:

```
ldefs -c interp e - 0 0 0
ldefs -c interp f - 0 0 0
```

2) Home the two channels, and reset the software counters for these two channels to zero:

```
! home e f
reset ef 0
```

3) Load a focus chart in the projector.

4) Use the viewer in the camera to verify sharp focus for the zero position.

   If it's not in focus at zero, you better find out why by fixing the HOMEDEFS.HOM file, so that zero for the zoom is also zero for the focus channel.

5) Now we move the zoom (e) to the extreme negative position:

```
go e >-40000
```

6) Use 'jog f' to jog the f channel until the focus chart is in focus. Write this focus position down for the current zoom position in the little table (described above).

7) Move the zoom (e) channel forward 10000 steps, and repeat steps 6 and 7 until you fill the table with focus positions for each zoom position.

8) Using your table of numbers, which is let's say:

ZOOM(E)	FOCUS(F)	
-40000	-212900	
-30000	-153500	
-20000	-96050	
-10000	-43150	
0	0	<-- we know this is zero already
10000	7800	
20000	13375	

9) Now edit the OPCSDEFS.OPC file, and create an 'interp' command down in the FADER AND FOCUS section that will configure these focus positions, so that moving the zoom will cause the focus channel to try to keep the projector in focus:

```
interp f e -40000 20000 7  -212900 -153500 -96050 -43150 0 7800 13375
      - - ------ ----- -  ----------------------------------------
      | |   |      |   |  These are focus positions from your table
      | |   |      |   |
      | |   |      |   |
      | |   |      |   The number of focus positions
      | |   |      |
      | |   |      End zoom position
      | |   |
      | |   Start zoom position
      | |
      | Zoom channel (lens)
      |
      Focus channel (camera base)
```

Make sure there's no other 'interp f e' command in the file.. if there is, remove it to prevent confusion.

10) Now reload the OPCSDEFS.OPC file so that the new interp command is configured by running: ldefs opcsdefs.opc

11) Check that focus is maintained by moving the zoom to various positions, and check focus. e.g.

go ef >25000
go ef >18000

Note you need to specify both channels for follow focus to work. If you just use 'go e >25000', that will just move the zoom without doing follow focus.

That's it.

If focus seems dodgy for the inbetween positions, you may need to double the number of focus positions by adjusting your zoom increment and repeating the above procedure.

So in the above example, instead of using 10000, use 5000 increments on the zoom, which will double the number of focus positions to find, making a tighter curve.

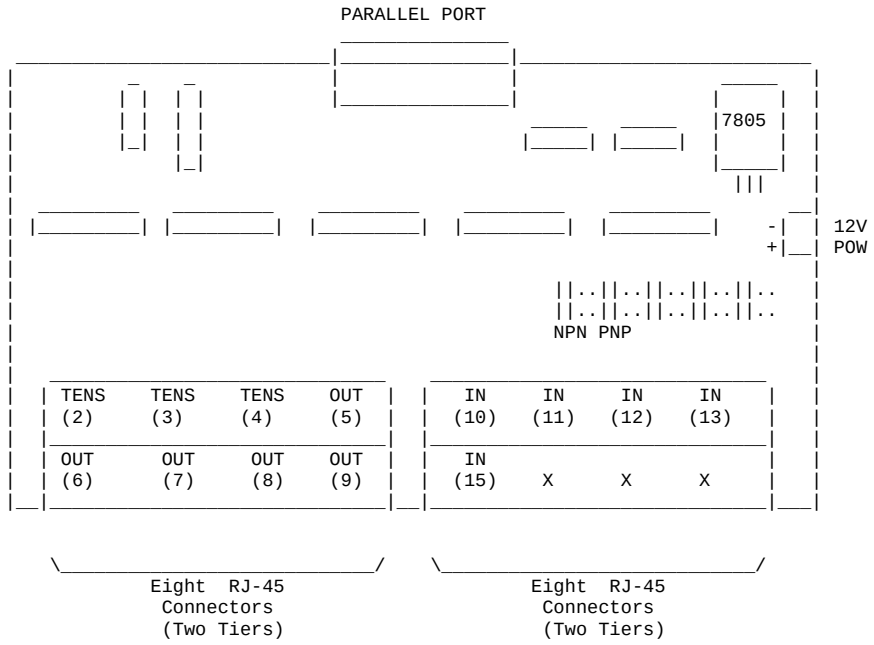
In many cases, I've seen needing 30 or 40 interpolation positions for accurate follow focus.

ORIGIN

Gregory Ercolano, Topanga, California 04/12/00

```
OPCSIFACE(DOCS)          Optical Printer Control System    OPCSIFACE(DOCS)

NAME
    OPCSIFACE - OPCS Interfacing docs

------------------------------------------------------------------------------
                    PIO-100 - PARALLEL PORT INTERFACING
------------------------------------------------------------------------------

NAME
    pio-100 - OPCS parallel port I/O interface board

DESCRIPTION
    The OPCS parallel port interface board (PIO-100) was designed to
    simplify wiring between the computer parallel port and the various
    digital sensors on the printer, using standard RJ-45 patch cables
    to route the signals to each sensor. The board also optically
    isolates the computer and the optical printer's digital sensors,
    namely home sensors, buckle/viewer switches, deenergize options,
    tension motors, etc.

    There are several revisions of this board:

        REV 3/Feb 2010: First use by Disney (YCM printers), used by others
                        See: http://seriss.com/opcs/docs/parallel-port-interface/rev3

        REV 6/Jan 2021: First use by Mike Ferriter, Andy Kaiser,
                        Bruce Heller, Carl Spencer, etc.
                        See: http://seriss.com/opcs/pio-100/

REV 6 "PIO-100" Parallel I/O Board - Jan 2021
=============================================
    This board has a webpage with schematics, wiring diagrams, PCB layouts,
    photos, and other useful information here:

        http://seriss.com/opcs/pio-100/

    As of this writing (Aug 2021), REV 6 is the latest revision of this board.
    This board was branded with the model number "PIO-100", to differentiate
    it from the other OPCS boards (A800, SD-800, etc).

    At the top, a parallel port connector is connected to the computer's
    parallel port via a DB-25 ribbon cable. On the right side, a single
    12V power connector. Derives 5V with an onboard 7805 used for the
    computer interface.

    While this board is optically isolated for the signals, there is a
    common ground between the 12V and 5V supplies.

    Along the bottom are 16 RJ-45 connectors arranged in two-tier
    connector blocks. These fan out to the optical printer's sensors
    and motor controls as individual RJ-45 patch cables, one per
    device. These devices can be 12V home sensors (or 'optical sensors'),
    tension motor control relays (SSR's), buckle/viewer switches,
    motor enable/disable controls, etc.
```

The REV 6 board looks like this:



Regarding the labels on the RJ-45 connectors,
the numbers in parentheses are the parallel port pin#s:

> Outputs (from the computer) are pins 2 thru 9.
> Inputs (to the computer) are pins 10 thru 13, and 15.

TENSION OUTPUTS

At the bottom left, there are three 'TENSION' outputs intended to control the SSR relays for tension motors, one RJ-45 output cable per pair of feed/takeup motors, one pair for each film movement, which is typically:

TENS(2) -- Aerial Projector (feed/takeup)
TENS(3) -- Main Projector (feed/takeup)
TENS(4) -- Camera (feed/takeup)

Changing a bit on one of these outputs inverts the state of the feed/takeup so that only one of the two tension motor relays is on, and the other off. In the OPCS software's setup file, OPCSDEFS.OPC, the TENSION(OPCSDEFS) command is used to configure this for each channel that supports tension motors.

When the channel is running forward, the TAKEUP motor is energized, and FEED is disabled. Typically a small high power low ohm rating resistor lies across each SSR relay's output, allows a small amount of 110VAC to run the tension motor as a "holding current" when the relay is off. When the relay is on, full 110 VAC drives the tension motor. Actual voltage to the motors are usually tunable with a variac the camera operator can set.

TENSION(2,3,4) OUTPUTS
RJ-45 PINOUTS

PIN#	DESCRIPTION	COLOR
1	GND	WHT/ORN
2	TAKEUP(-)	ORN
3	GND	WHT/GRN
4	TAKEUP(+)	BLU
5	GND	WHT/BLU
6	FEED(-)	GRN
7	GND	WHT/BRN
8	FEED(+)	BRN

GENERIC OUTPUTS

Since the first three output pins of the parallel port are used for tension motors, the remaining five pins are generic optically isolated 12V outputs that can be used for various purposes. Often these are used to deenergize channels, allowing the software to unlock motor(s) on command, allowing the operator to freewheel the motor, then the software can re-home the motor on completion.

Generic output control can be done via the 'home' command as configured in the HOMEDEFS.HOM file, using either the 'setbit' or 'clrbit' commands. Similar commands in the OPCSDEFS.OPC file and/or OPCS run scripts can be used to change the parallel port's bits via command control, e.g.

ldefs -c setbit 0378 8 0 -- set parallel port pin #5 (bitmask 0x08)
ldefs -c clrbit 0378 8 0 -- clear parallel port pin #5 (bitmask 0x08)

OUT(5,6,7,8,9) OUTPUTS
RJ-45 PINOUTS

PIN#	DESCRIPTION	COLOR	
1	GND	WHT/ORN	
2	GND	ORN	
3	GND	WHT/GRN	
4	N/C	BLU	
5	GND	WHT/BLU	
6	OUTPUT	GRN	<-- LOW=GND HI=+12V
7	GND	WHT/BRN	
8	+12	BRN	

GENERIC INPUTS

The generic inputs IN(10) thru IN(13) and IN(15) can be used for either home sensors, buckle/viewer switches, etc. These respond to voltages typically 12V (for "on") or pulled to Ground (for off).

+12 and Ground signals are provided on each RJ-45 port to be used for driving the home sensor's internal circuits and for 12v/Gnd reference.

Home sensors are typically configured for the 'home' command using the HOMEDEFS.HOM file's 'homeport' command, which procedures in that file can then use to test the home sensor to conditionally run motors.

Buckle and Viewer switches can also be used to drive these inputs.

Schematics are available on the website, and also are printed on the board's silk screen for reference, along with simple wiring diagrams.

### IN(10,11,12,13,15) ###		
### RJ-45 PINOUTS ###		
PIN#	DESCRIPTION	COLOR
1	GND	WHT/ORN
2	GND	ORN
3	GND	WHT/GRN
4	N/C	BLU
5	GND	WHT/BLU
6	IN	GRN
7	GND	WHT/BRN
8	+12	BRN

INPUT JUMPERS

To support both NPN and PNP home sensors, a jumper block is provided on the board to allow either type to be supported. The default is NPN, which is the most common sensor type. It is advised you standardize on only one type of sensor for all sensors, so they can be easily reassigned without having to change the jumpers.

WARNING: BE SURE THE BOARD'S 12V POWER IS REMOVED BEFORE CHANGING JUMPERS.
If you must change the jumpers while the board is "hot", remove *both jumpers completely* before replacing to the new positions. AVOID changing one jumper at a time, as that can short the 12V power supply during mid-change.

CAVEATS

The RJ-45 connectors labeled "X" are unused for I/O, but can be used for access to +12V and GND from the board for various purposes (such as 12V power lights, etc)

On the REV 6 board, there are a TWO MINOR ERRORS that will be fixed in future revisions (probably REV 6A and up):

> Many of the little diagrams on the silk screen are wrong. White labels are affixed over these problem diagrams to make corrections. All REV 6 boards in the field should already have these white 'fix labels' on them.

> Two of the outputs, OUT(8) and OUT(9), do not match the normal wiring pattern of the other connectors. It's advised you do not use OUT(8) and OUT(9) on the REV 6 board, for consistency.

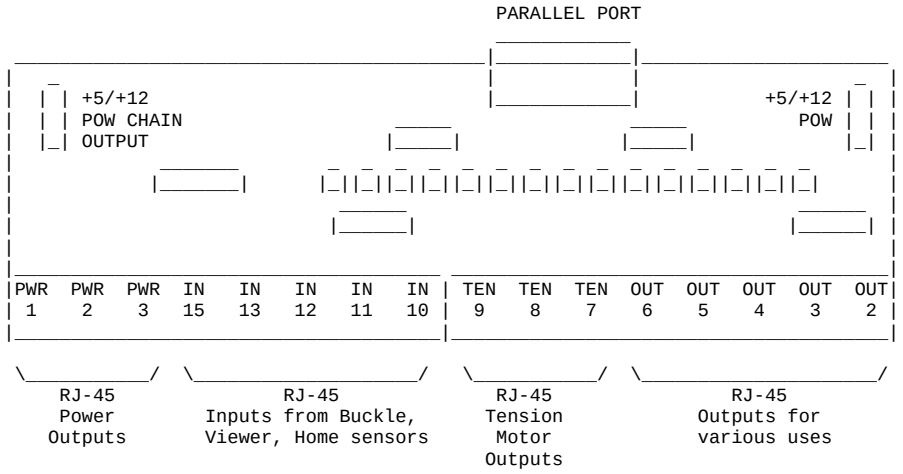
REV 3 "Parallel Port Interface Board" - Feb 2010
==========

This board has a webpage with schematics, wiring diagrams, PCB layouts, photos, and other useful information here:

http://seriss.com/opcs/docs/parallel-port-interface/rev3/

The REV 3 board uses separate +5V and +12V power, to ensure complete isolation. But it is possible to use a single dual +5v/+12v power supply and share the signal ground.

At the top, a parallel port connector is connected to the computer's parallel port via a ribbon cable. On the sides, power connectors for the input +12V and +5V. Along the bottom, RJ-45 connectors are used to fan out to the optical printer's sensors and motor controls; home sensors, tension motors, buckle/viewer switches, motor enable/disable controls, etc. It looks like this:



For the most part, the buckle/viewer sensors are configured by the BUCKLE(OPCSDEFS) and VIEWER(OPCSDEFS) commands in the OPCSDEFS.OPC file to define the port and bit mask values corresponding to the RJ-45 ports used for those features.

The home sensors are configured in the HOME(DOCS) program's HOMEDEFS.HOM to define the port and bit mask values corresponding to the RJ-45 ports used for those features.

The tension motor controls are configured with the TENSION(OPCSDEFS) command in the OPCSDEFS.OPC file to define the port and bit mask values corresponding to the RJ-45 ports used for those features, and are wired specially with Crydom solid state relays to control the AC tension motors.

Various other inputs/outputs can be controlled by these ports, such as energizing/deenergizing certain motors via OPCS commands. An example would be the LOAD command, which might run the motors by small amounts to unseat the film, before deenergizing for manual adjustment.

PARALLEL CONNECTOR

The parallel connector on the OPCS parallel port interface board is a female DB-25 connector, which should be connected to one of the computer's parallel ports.

PIN	PORT	MASK	I/O	RJ-45	DESCRIPTION
2	0x378	0x01	Out	OUT(2)	Generic output
3	0x378	0x02	Out	OUT(3)	Generic output
4	0x378	0x04	Out	OUT(4)	Generic output
5	0x378	0x08	Out	OUT(5)	Generic output
6	0x378	0x10	Out	OUT(6)	Generic output
7	0x378	0x20	Out	OUT(7)	Generic output
8	0x378	0x40	Out	TEN(8)	Camera Tension
9	0x378	0x80	Out	TEN(9)	Projector Tension
10	0x379	!0x40	In	IN(10)	Generic Input
11	0x379	!0x80	In	IN(11)	Generic Input
12	0x379	0x20	In	IN(12)	Generic Input
13	0x379	0x10	In	IN(13)	Generic Input
15	0x379	0x08	In	IN(15)	Generic Input
18-25	-	-	Gnd	-	Ground

RJ-45 CONNECTORS

INPUTS - IN(10-15)

The 5 generic inputs are real time inputs that can be read by the computer. The OPCS software can be configured to make use of these inputs by specifying the corresponding port/mask via the OPCSDEFS.OPC or HOMEDEFS.HOM files.

Typically generic inputs are used for either home sensors or buckle/viewer switch sensing.

IN(10) - IN(15)
RJ-45 PINOUTS

PIN#	DESCRIPTION	COLOR
1	Chassis	WHT/ORN
2	GND	ORN
3	Chassis	WHT/GRN
4	-	BLU
5	Chassis	WHT/BLU
6	IN	GRN
7	Chassis	WHT/BRN
8	+12	BRN

OUTPUTS - OUT(2-7)

The 6 generic outputs can be controlled directly by commands in HOMEDEFS.HOM or OPCSDEFS.OPC, e.g. the SETBIT, CLRBIT, and XORBIT commands.

Typically, generic outputs are used for deenergizing motors to allow manual load/unload of film with the custom LOAD and LINEUP commands.

OUT(2) - OUT(7)
RJ-45 PINOUTS

PIN#	DESCRIPTION	COLOR
1	Chassis	WHT/ORN
2	GND	ORN
3	Chassis	WHT/GRN
4	-	BLU
5	Chassis	WHT/BLU
6	OUT	GRN
7	Chassis	WHT/BRN
8	+12	BRN

TENSION OUTPUTS - TEN(8) AND TEN(9)

The tension motor outputs TEN(8) and TEN(9) can control the FEED and TAKEUP motors for camera and projector.

When parallel port pin 8's bit changes from 0 to 1, the TEN(8) RJ-45 connector's FEED and TAKEUP outputs will change state, always being the compliment of each other (ie. if FEED is 'on', TAKEUP will be 'off').

### TEN(8) AND TEN(9) RJ-45 PINOUTS ###			
PIN#	DESCRIPTION	COLOR	CRYDOM PIN#
1	Chassis	WHT/ORN	4
2	-TAKEUP	ORN	-
3	Chassis	WHT/GRN	3
4	+TAKEUP	BLU	-
5	Chassis	WHT/BLU	4
6	-FEED	GRN	-
7	Chassis	WHT/BRN	3
8	+FEED	BRN	-

POWER OUTPUTS - PWR(1) THRU PWR(3)

PWR-1 through PWR-3 can be used to supply +12V power to the printer.

### PWR-1 THRU PWR-3 RJ-45 PINOUTS ###			
PIN#	DESCRIPTION	COLOR	CRYDOM PIN#
1	Chassis	WHT/ORN	4
2	GND	ORN	-
3	Chassis	WHT/GRN	3
4	-	BLU	-
5	Chassis	WHT/BLU	4
6	-	GRN	-
7	Chassis	WHT/BRN	3
8	+12	BRN	-

SD-800 - STEPPER DISTRIBUTION INTERFACE

NAME

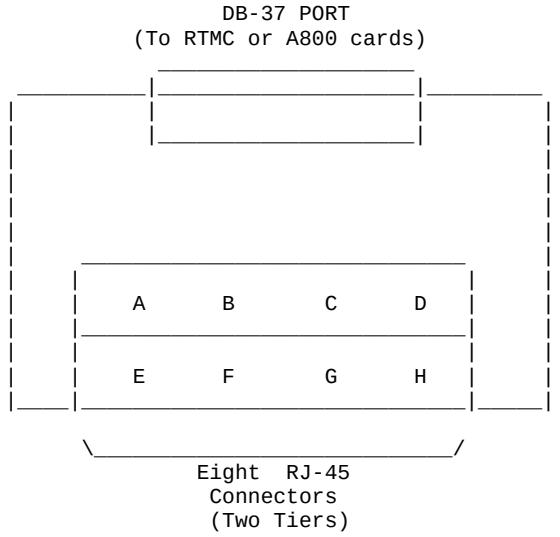
sd-800 - OPCS 8 channel "stepper distribution" (SD) card

DESCRIPTION

The OPCS "Stepper Distribution" card (SD-800) was designed to simplify wiring between the computer step pulse generator card (e.g. RTMC16, RTMC48, Kuper Industrial, A800..) and the stepper motor driver modules (Centent, Gecko, LeadShine, etc) by breaking out the DB-37 connector into separate RJ-45 patch cables, one per stepper drive channel.

This board really has no active features on it, other than a fanout to simplify wiring. Optional pullup resistor networks can be used if the application requires open collector outputs from the card to be pulled up to +5V for the idle state to prevent noise.

As of this writing, there is only one version of the board, REV 0, which looks like this:



Typically the female DB-37 connector on the board is connected to the DB-37 connector on the ISA stepper pulse generator card plugged into the the DOS computer using 6' male/male cable.

And separate RJ-45 patch cables are wired to the A/B/C/D.. ports at the bottom of the board, which run out to the individual stepper drives (Centent, Gecko, LeadShine, etc).

The DB-37 follows Kuper's pinout; see 'man kuper' for more info.
The RJ-45 pinout diagram is on the board, but is basically:

	RJ-45 PIN#	SIGNAL	WIRE COLOR (*)	CENTENT DRIVE	GECKO DRIVE	LEADSHINE DRIVE
	1	GND	-	N/C	N/C	N/C
	2	GND	-	N/C	N/C	N/C
	3	GND	-	N/C	N/C	N/C
DIR	4	DIRECTION	BLU	DIRECTION	(8) DIR	DIR-(DIR)
	5	+5V	WHT/BLU	+5 VOLTS DC	(10) COMMON	DIR+(5V-24V)
	6	GND	-	N/C	N/C	N/C
STP	7	+5V	WHT/BRN	N/C	N/C	PUL+(5V-24V)
	8	STEPS	BRN	STEP PULSE	(9) STEP	PUL-(PUL)

(*) Premade RJ-45 patch cables for cat5 and cat5e usually have the standard wire colors shown above. For the signals used, the wiring colors are the same for 568A and 568B.

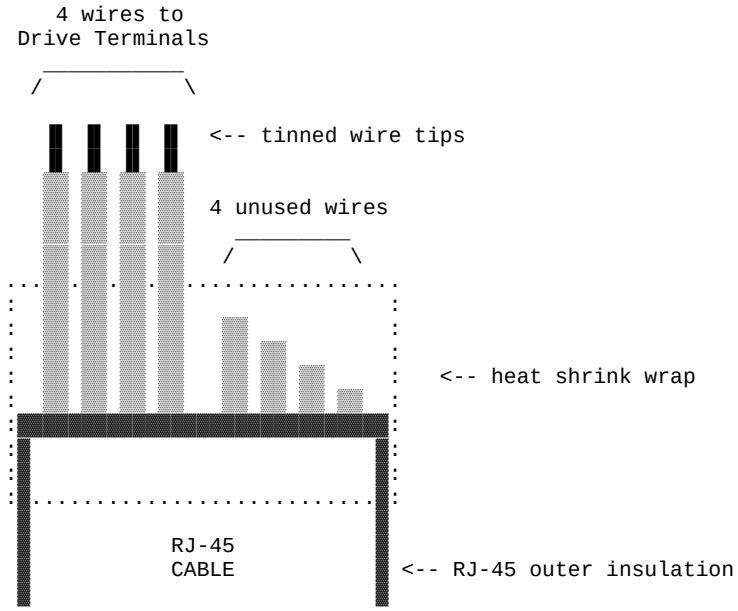
Basically only 4 of the 8 wires are used. In some cases only 3 wires are used (Centent & Gecko).

Please note these signals are DIRECTLY FROM THE COMPUTER MOTHERBOARD, so be very careful with them. Do not let them short to chassis ground on the printer, or to each other.

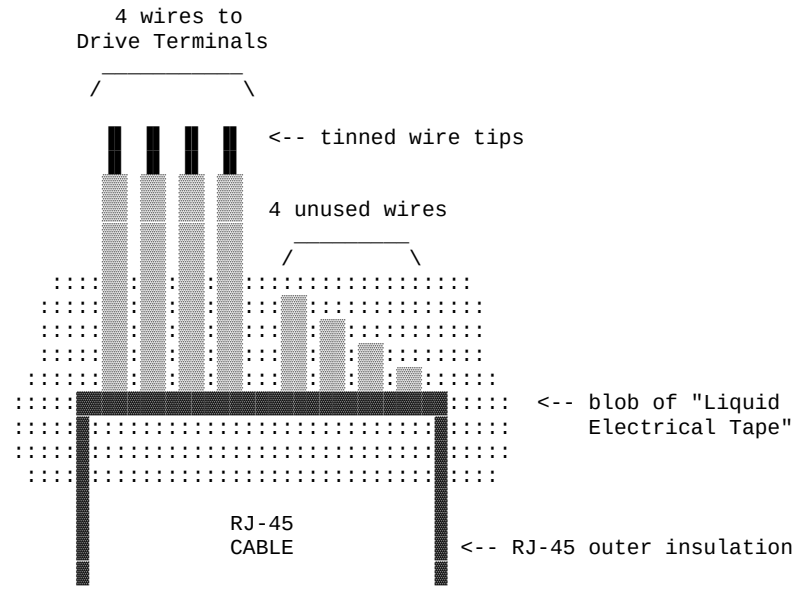
RJ-45 CABLE TERMINATION

For unused (N/C) wires, be sure to isolate them from each other to prevent shorts. Cut them short to the stripped insulation or stagger the lengths of the unused wires to prevent the frayed ends from shorting into each other.

Wrap the unused conductors with a small piece of heat shrink:



If heat shrink is not available, a blob of black liquid electrical tape at the end of the cables outer insulation can be used instead:



Be sure the ends of the unused wires are well isolated from the open air, and from each other during drying.

ISOLATE UNUSED RJ-45 CONDUCTORS

Unused conductors on the SD-800 and SD-1600 are usually just the chassis grounds, and are not critical to isolate from each other.

However, unused conductors on the PIO-800 Parallel Input/Output boards are typically +5V and GND from the computer, and therefore should be /carefully isolated/ from each other.

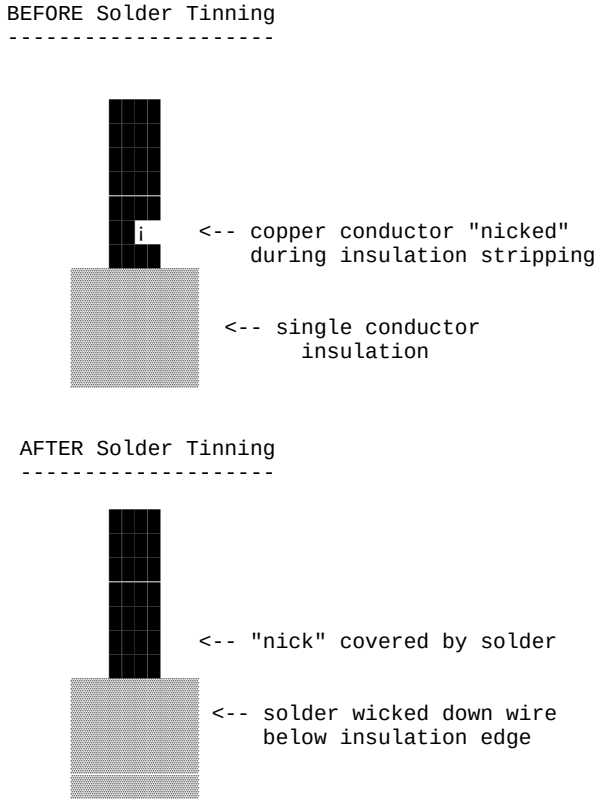
When wiring SD-800 and SD-1600 boards to the Centent or Gecko drives, strip the two +5V signal wires and twist them together before tinning with solder. Don't let one of the +5V signal wires hang loose, because if it touches anything else, it will short the computer's +5V supply, which at best reboots the machine, or at worst toasts the computer's mother board.

STRIPPING RJ-45 CABLES

When wiring to the screw clamp terminals, tin the wires (if they're stranded) before inserting them, to prevent wire fraying and shorts from stray pieces of stranded wire.

When tinning stranded wire, first twist the strands into a spiral, then dip the twisted strands into some flux. This way when solder is applied, the extra flux helps the solder wick down the conductors beneath the insulation, stiffening the conductors beyond insulation's edge, preventing flexion at the end of the wire.

Tinning will also cover over any nicks in the conductors that may have occurred during wire stripping with solder, stiffening the nicks, preventing fatigue:



RJ-45 STRAIN RELIEF

Don't let the cables hang by their screw terminals at motor drives and/or termination at home sensors. Casual wire flexion will cause wire fatigue, leading to breakage or intermittent signal dropouts.

Secure cables to the chassis or drive units by their insulation within approx 6" of the wire termination point using zip ties or nylon straps/cleats.

A800 - STEP PULSE GENERATOR INTERFACE

NAME

A800 - Seriss Corp. A800 stepper motor control card

DESCRIPTION

The A800 card is a "short slot" ISA card for the IBM PC that can generate steps/direction pulse streams to control up to 8 stepper motors at once.

The card uses two PIC chips to manage the stepper pulse generation. The PIC's firmware and MS-DOS driver "A800DRV.COM" source code are open source and available from:

https://github.com/erco77/a800-opcs-pic-asm

OPCS communicates with the A800 card by way of the MS-DOS device driver "A800DRV.COM", which provides a standard low level interface to the card that OPCS can make use of to run the motors efficiently.

The A800DRV.COM driver must be loaded *before* running the OPCS software. This can be installed either by the AUTOEXEC.BAT, or by a separate batch script that invokes OPCS.

If the A800 card's jumpers are default (BaseAddr=300 and IRQ=5), then you can install the driver with just:

a800drv

CONFIGURING THE BASEADDR AND IRQ

In OPCS K1.xx, the a800 card did not exist and is not supported.

In OPCS K2.00 through K2.09, the base address is configured in OPCSDEFS.OPC with the 'baseaddr' command. IRQ not configurable.

In OPCS K2.10 and up, the A800DRV.COM driver allows both the base address and IRQ to be configured on the command line. The default would be:

```
a800drv -b300 -i5           <-- Sets base address=0300h, IRQ=5
           |    |
           |    IRQ=5
           Base Addr=300
```

..and if your A800 jumpers are set differently, then specify matching values accordingly. e.g. if the card's jumpers are set to BaseAddr=340 and IRQ=6, then start the driver with:

```
a800drv -b340 -i6
```

To list the A800DRV driver's options, run 'a800drv -help'. If it does not show a list of options, then it is an older version that does not support command line options.

TECHNICAL SYNOPSIS

When the software wants to move a motor, it provides 8 separate 12 bit velocity values, one per motor channel. And 107 of these velocity values are sent per second to the card using the hardware interrupt on IRQ 5.

Currently only 8 bits of the 12bit value are used for motor speeds. i.e. the lowest velocity is 1 (107 Hz) and the highest velocity is 255 (27,285 Hz). Values above 255 are clipped by the hardware, as the PIC chips are limited by their speed. The high bit (0x8000) is the motor direction bit; 0=forward, 1=reverse.

The software has to keep up with this transmission rate, otherwise it will lose track of the motor positions. The A800DRV.COM device driver provides a 64k ring buffer for the motor velocities that OPCS updates in real time while the motors are running.

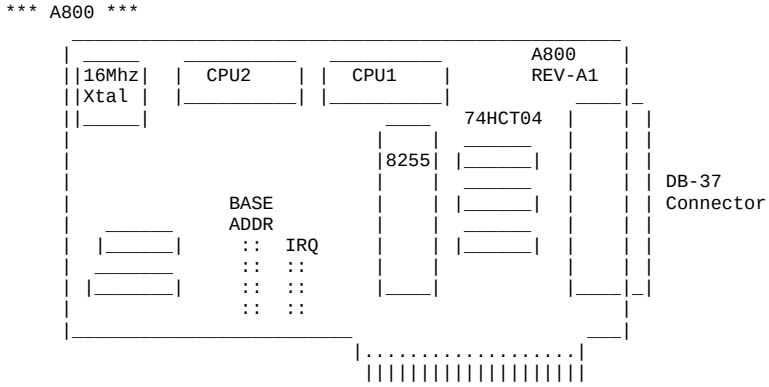
The OPCS software and A800DRV.COM use INT 99h to intercommunicate, providing the address of the ring buffer, and start/stop commands.

The A800 card generates 107 interrupts per second to the A800DRV.COM driver, each interrupt feeds 8 velocities from the tail of the ring buffer to the A800 card, and increments the tail's index to point to the next 8 values in the ring buffer. Meanwhile, the OPCS software feeds velocities into the head of the ring buffer, always keeping ahead of the tail. If the tail catches up to the head prematurely, this causes a SYNC FAULT error, which should never happen unless something is wrong with the computer.

OPCS A800 CARD

==========

This card controls 8 axes and is a half sized IBM PC ISA card.
For complete info on this card, see: http://seriss.com/opcs/a800



DB-37 Connector (similar to Kuper):

PIN#	SIGNAL	PIN	SIGNAL
1	N/C	20	+5VDC
2	STEP A	21	DIR A
3	STEP B	22	DIR B
4	STEP C	23	DIR C
5	STEP D	24	DIR D
6	STEP E	25	DIR E
7	STEP F	26	DIR F
8	STEP G	27	DIR G
9	STEP H	28	DIR H
10	N/C	29	N/C
11	N/C	30	N/C
12	N/C	31	N/C
13	N/C	32	N/C
14	N/C	33	N/C
15	N/C	34	N/C
16	N/C	35	N/C
17	N/C	36	N/C
18	N/C	37	N/C
19	GND(*)		

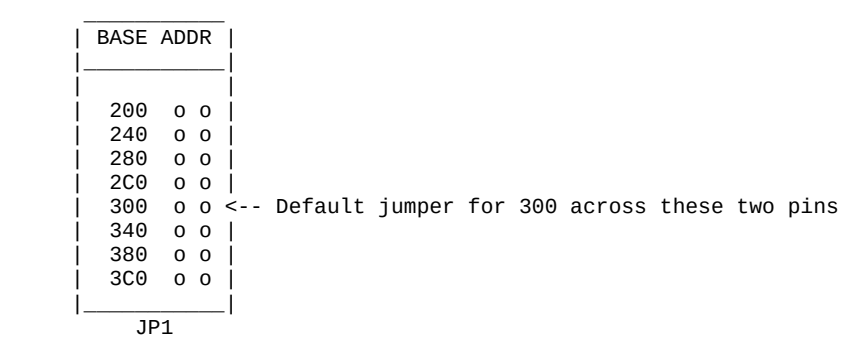
(*) = JP3 configures DB37 Pin#19:
"+5" - Makes Pin #19 +5 VDC
"GND" - Makes Pin #19 GND (default)

NOTE: When fitted with 74LS07 chips, outputs are OPEN COLLECTOR TTL.

When those chips are replaced with 74ALS1034N, outputs swing a full +5/GND and are CMOS/TTL compatible.

BASE ADDRESS (JP1)

Closeup of the 'BASE ADDRESS' jumpers (JP1), which sets the base address of the 8255 chip's I/O port registers:



A800 Base Address Jumpers

Always defer to the board's labeling (if any), as the board designs may have changed since this document's writing (May 2020).

DEFAULTS:

This board has labels for the BASE ADDRESS and IRQs:
"300" is the default base address (5th pair of pins from top jumpered).
"IRQ5" is the default IRQ (4th pair of pins from top jumpered).

DB-37 OUTPUT SIGNALS

The STEPS output are normally high (+5) during idle, and fall low (GND) to pulse the motor a single step.

The outputs for DIR (direction) are logic hi (+5) for forward, and logic low (GND) for reverse.

The output signals can either be CMOS hi/low levels, or can be "open collector" (where logic 'hi' is 'open', and logic low is gnd). Which it is depends on the chips installed in the three chip positions to the left of the DB-37 connector on the A800 board:

```
74HCT04 -- CMOS high/low levels (default)
74LS07  -- Open Collector
```

For controlling the modern DM542 and FMD27400 motor drivers, the 74HCT04 chips are recommended in these positions.

For Centent and Gecko drives, traditionally 74LS07 chips were used, but will probably also work with the 74HCT04's.

While both chips work on all drives, analysis with an oscilloscope monitoring the stepper drive inputs may reveal one chip is better than the other for noise reduction. With 6' cables, 74HCT04 seems the best choice.

Always defer to the board's silk screen labelling, as the board designs may change since this document's writing (May 2020).

HISTORY

Greg Ercolano designed this card in May/June 2020, and the driver software, A800DRV.COM. This card uses "PIC chips", which are programmed with firmware written in the processor's native assembly language for speed and consistent timing for generating the steps and direction motor signals.

SEE ALSO

RTMC16(DOCS) - notes on the Kuper Controls RTMC16 motor control card
RTMC48(DOCS) - notes on the Kuper Controls RTMC48 motor control card
8255(DOCS) - how to control 8255 based digital I/O cards
KUPER(DOCS) - documentation on the kuper card connectors
SD-800(DOCS) - 8 channel stepper distribution board (simplify DB-37 wiring)
SD-1600(DOCS) - 16 channel stepper distribution board (simplify DB-37 wiring)

AUTHOR

Greg Ercolano / Seriss Corporation 2021

PARALLEL(DOCS) Optical Printer Control System PARALLEL(DOCS)

NAME

parallel - parallel port pinout and DOS monitoring tool

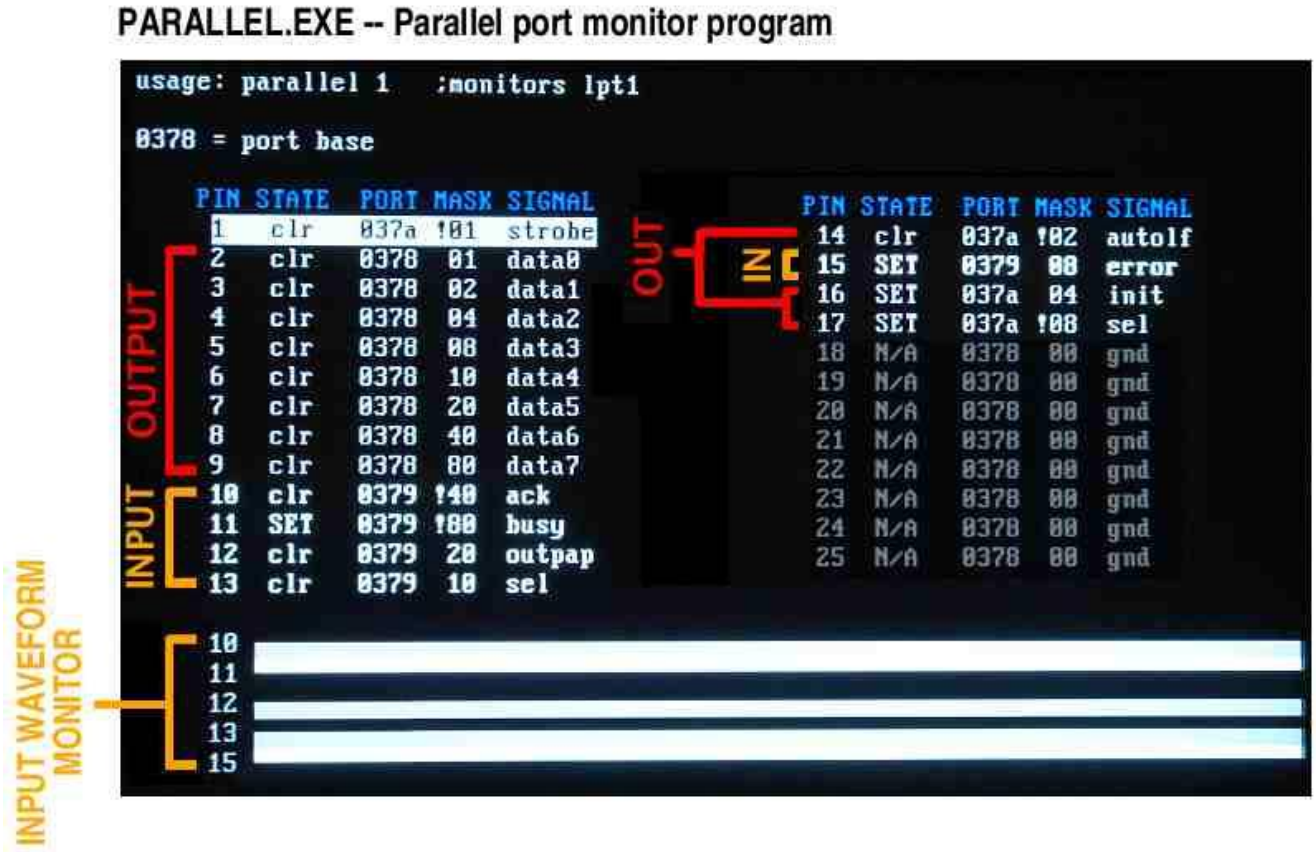
PARALLEL PORT PINOUT CHART

PIN	I/O	NAME	PORT	MASK (hex)
1	out	strobe	3be/37a	!01
2	out	data0	3bc/378	01
3	out	data1	3bc/378	02
4	out	data2	3bc/378	04
5	out	data3	3bc/378	08
6	out	data4	3bc/378	10
7	out	data5	3bc/378	20
8	out	data6	3bc/378	40
9	out	data7	3bc/378	80
10	in	acknow	3bd/379	40
11	in	busy	3bd/379	!80
12	in	out of pap	3bd/379	20
13	in	select	3bd/379	10
14	out	autofeed	3be/37a	!02
15	in	error	3bd/379	08
16	out	init	3be/37a	04
17	out	select	3be/37a	!08
18-25	ground	ground	-	-

PARALLEL PORT MONITOR PROGRAM

The OPCS software comes with parallel.exe, a program that monitors the real time status of the IBM PC's parallel ports. Run 'parallel.exe'.

SCREENSHOT:



```
USAGE
    parallel [-h] [port|lpt#]

EXAMPLES
   parallel         - monitor LPT1 (default)
   parallel 1       - monitor LPT1
   parallel 2       - monitor LPT2
   parallel 3       - monitor LPT3
   parallel 378     - monitor parallel port at base address 0378h
   parallel -h[elp] - help screen

KEYS
    UP/DOWN    - move edit cursor up/down
    ENTER      - toggles state of output (when cursor on an output)
    ESC        - quit program

    While the edit cursor is positioned on an input, the speaker makes
    a 3000 HZ tone if the input is HIGH, and makes no sound if LOW.

SOURCE
    The GPL3 Turbo-C source code for the parallel port program can be
    found at: http://github.com/erco77/parallel-dos

    The binary "parallel.exe" can be downloaded from:
    http://seriss.com/opcs/ftp/

ORIGIN
    Gregory Ercolano, Los Feliz California xx/xx/1988
```

PIO-100(DOCS) Optical Printer Control System PIO-100(DOCS)

NAME

pio-100 - OPCS parallel port I/O interface board

DESCRIPTION

The OPCS parallel port interface board (PIO-100) was designed to simplify wiring between the computer parallel port and the various digital sensors on the printer, using standard RJ-45 patch cables to route the signals to each sensor. The board also optically isolates the computer and the optical printer's digital sensors, namely home sensors, buckle/viewer switches, deenergize options, tension motors, etc.

There are several revisions of this board:

REV 3/Feb 2010: First use by Disney (YCM printers), used by others
See: http://seriss.com/opcs/docs/parallel-port-interface/rev3

REV 6/Jan 2021: First use by Mike Ferriter, Andy Kaiser, Bruce Heller, Carl Spencer, etc.
See: http://seriss.com/opcs/pio-100/

REV 6 "PIO-100" Parallel I/O Board - Jan 2021

This board has a webpage with schematics, wiring diagrams, PCB layouts, photos, and other useful information here:

http://seriss.com/opcs/pio-100/

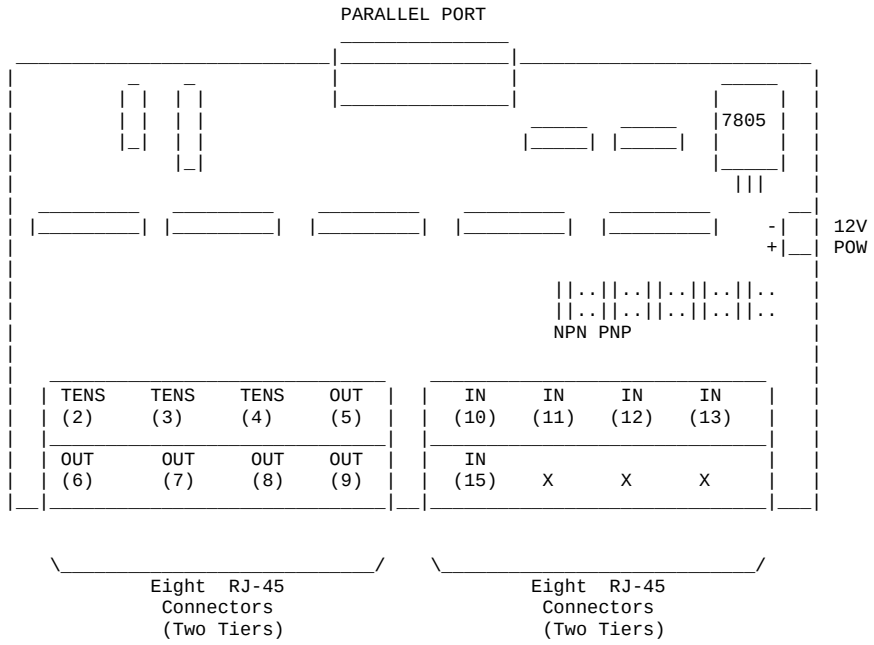
As of this writing (Aug 2021), REV 6 is the latest revision of this board. This board was branded with the model number "PIO-100", to differentiate it from the other OPCS boards (A800, SD-800, etc).

At the top, a parallel port connector is connected to the computer's parallel port via a DB-25 ribbon cable. On the right side, a single 12V power connector. Derives 5V with an onboard 7805 used for the computer interface.

While this board is optically isolated for the signals, there is a common ground between the 12V and 5V supplies.

Along the bottom are 16 RJ-45 connectors arranged in two-tier connector blocks. These fan out to the optical printer's sensors and motor controls as individual RJ-45 patch cables, one per device. These devices can be 12V home sensors (or 'optical sensors'), tension motor control relays (SSR's), buckle/viewer switches, motor enable/disable controls, etc.

The REV 6 board looks like this:



Regarding the labels on the RJ-45 connectors,
the numbers in parentheses are the parallel port pin#s:

> Outputs (from the computer) are pins 2 thru 9.
> Inputs (to the computer) are pins 10 thru 13, and 15.

TENSION OUTPUTS

At the bottom left, there are three 'TENSION' outputs intended to control the SSR relays for tension motors, one RJ-45 output cable per pair of feed/takeup motors, one pair for each film movement, which is typically:

TENS(2) -- Aerial Projector (feed/takeup)
TENS(3) -- Main Projector (feed/takeup)
TENS(4) -- Camera (feed/takeup)

Changing a bit on one of these outputs inverts the state of the feed/takeup so that only one of the two tension motor relays is on, and the other off. In the OPCS software's setup file, OPCSDEFS.OPC, the TENSION(OPCSDEFS) command is used to configure this for each channel that supports tension motors.

When the channel is running forward, the TAKEUP motor is energized, and FEED is disabled. Typically a small high power low ohm rating resistor lies across each SSR relay's output, allows a small amount of 110VAC to run the tension motor as a "holding current" when the relay is off. When the relay is on, full 110 VAC drives the tension motor. Actual voltage to the motors are usually tunable with a variac the camera operator can set.

TENSION(2,3,4) OUTPUTS
RJ-45 PINOUTS

PIN#	DESCRIPTION	COLOR
1	GND	WHT/ORN
2	TAKEUP(-)	ORN
3	GND	WHT/GRN
4	TAKEUP(+)	BLU
5	GND	WHT/BLU
6	FEED(-)	GRN
7	GND	WHT/BRN
8	FEED(+)	BRN

GENERIC OUTPUTS

Since the first three output pins of the parallel port are used for tension motors, the remaining five pins are generic optically isolated 12V outputs that can be used for various purposes. Often these are used to deenergize channels, allowing the software to unlock motor(s) on command, allowing the operator to freewheel the motor, then the software can re-home the motor on completion.

Generic output control can be done via the 'home' command as configured in the HOMEDEFS.HOM file, using either the 'setbit' or 'clrbit' commands. Similar commands in the OPCSDEFS.OPC file and/or OPCS run scripts can be used to change the parallel port's bits via command control, e.g.

ldefs -c setbit 0378 8 0 -- set parallel port pin #5 (bitmask 0x08)
ldefs -c clrbit 0378 8 0 -- clear parallel port pin #5 (bitmask 0x08)

OUT(5,6,7,8,9) OUTPUTS
RJ-45 PINOUTS

PIN#	DESCRIPTION	COLOR	
1	GND	WHT/ORN	
2	GND	ORN	
3	GND	WHT/GRN	
4	N/C	BLU	
5	GND	WHT/BLU	
6	OUTPUT	GRN	<-- LOW=GND HI=+12V
7	GND	WHT/BRN	
8	+12	BRN	

GENERIC INPUTS

The generic inputs IN(10) thru IN(13) and IN(15) can be used for either home sensors, buckle/viewer switches, etc. These respond to voltages typically 12V (for "on") or pulled to Ground (for off).

+12 and Ground signals are provided on each RJ-45 port to be used for driving the home sensor's internal circuits and for 12v/Gnd reference.

Home sensors are typically configured for the 'home' command using the HOMEDEFS.HOM file's 'homeport' command, which procedures in that file can then use to test the home sensor to conditionally run motors.

Buckle and Viewer switches can also be used to drive these inputs.

Schematics are available on the website, and also are printed on the board's silk screen for reference, along with simple wiring diagrams.

IN(10,11,12,13,15)
RJ-45 PINOUTS

PIN#	DESCRIPTION	COLOR
1	GND	WHT/ORN
2	GND	ORN
3	GND	WHT/GRN
4	N/C	BLU
5	GND	WHT/BLU
6	IN	GRN
7	GND	WHT/BRN
8	+12	BRN

INPUT JUMPERS

To support both NPN and PNP home sensors, a jumper block is provided on the board to allow either type to be supported. The default is NPN, which is the most common sensor type. It is advised you standardize on only one type of sensor for all sensors, so they can be easily reassigned without having to change the jumpers.

WARNING: BE SURE THE BOARD'S 12V POWER IS REMOVED BEFORE CHANGING JUMPERS.
If you must change the jumpers while the board is "hot", remove *both jumpers completely* before replacing to the new positions. AVOID changing one jumper at a time, as that can short the 12V power supply during mid-change.

CAVEATS

The RJ-45 connectors labeled "X" are unused for I/O, but can be used for access to +12V and GND from the board for various purposes (such as 12V power lights, etc)

On the REV 6 board, there are a TWO MINOR ERRORS that will be fixed in future revisions (probably REV 6A and up):

> Many of the little diagrams on the silk screen are wrong. White labels are affixed over these problem diagrams to make corrections. All REV 6 boards in the field should already have these white 'fix labels' on them.

> Two of the outputs, OUT(8) and OUT(9), do not match the normal wiring pattern of the other connectors. It's advised you do not use OUT(8) and OUT(9) on the REV 6 board, for consistency.

REV 3 "Parallel Port Interface Board" - Feb 2010
==========

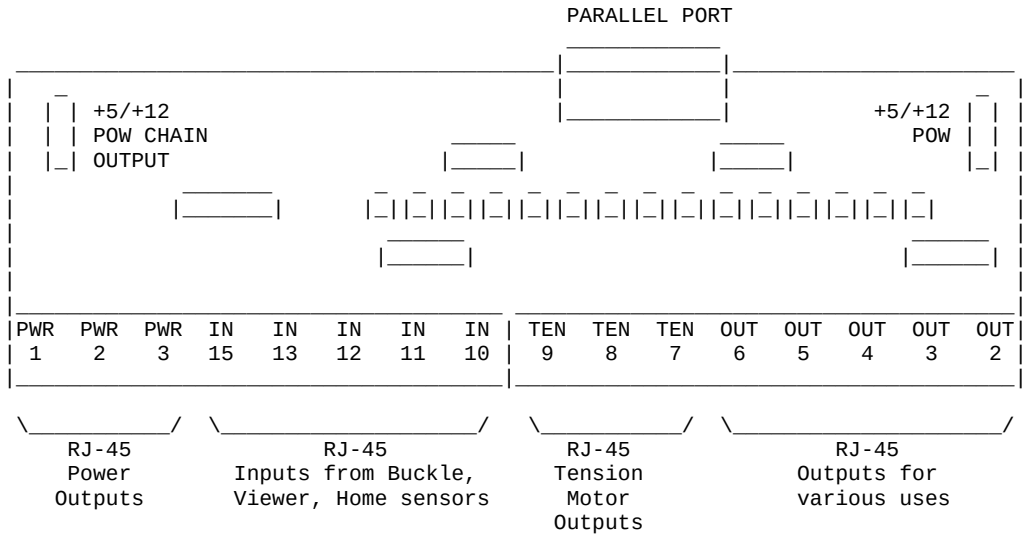
This board has a webpage with schematics, wiring diagrams, PCB layouts, photos, and other useful information here:

http://seriss.com/opcs/docs/parallel-port-interface/rev3/

The REV 3 board uses separate +5V and +12V power, to ensure complete isolation. But it is possible to use a single dual +5v/+12v power supply and share the signal ground.

At the top, a parallel port connector is connected to the computer's

parallel port via a ribbon cable. On the sides, power connectors for the input +12V and +5V. Along the bottom, RJ-45 connectors are used to fan out to the optical printer's sensors and motor controls; home sensors, tension motors, buckle/viewer switches, motor enable/disable controls, etc. It looks like this:



For the most part, the buckle/viewer sensors are configured by the BUCKLE(OPCSDEFS) and VIEWER(OPCSDEFS) commands in the OPCSDEFS.OPC file to define the port and bit mask values corresponding to the RJ-45 ports used for those features.

The home sensors are configured in the HOME(DOCS) program's HOMEDEFS.HOM to define the port and bit mask values corresponding to the RJ-45 ports used for those features.

The tension motor controls are configured with the TENSION(OPCSDEFS) command in the OPCSDEFS.OPC file to define the port and bit mask values corresponding to the RJ-45 ports used for those features, and are wired specially with Crydom solid state relays to control the AC tension motors.

Various other inputs/outputs can be controlled by these ports, such as energizing/deenergizing certain motors via OPCS commands. An example would be the LOAD command, which might run the motors by small amounts to unseat the film, before deenergizing for manual adjustment.

PARALLEL CONNECTOR

The parallel connector on the OPCS parallel port interface board is a female DB-25 connector, which should be connected to one of the computer's parallel ports.

PIN	PORT	MASK	I/O	RJ-45	DESCRIPTION
2	0x378	0x01	Out	OUT(2)	Generic output
3	0x378	0x02	Out	OUT(3)	Generic output
4	0x378	0x04	Out	OUT(4)	Generic output
5	0x378	0x08	Out	OUT(5)	Generic output
6	0x378	0x10	Out	OUT(6)	Generic output
7	0x378	0x20	Out	OUT(7)	Generic output
8	0x378	0x40	Out	TEN(8)	Camera Tension
9	0x378	0x80	Out	TEN(9)	Projector Tension
10	0x379	!0x40	In	IN(10)	Generic Input
11	0x379	!0x80	In	IN(11)	Generic Input
12	0x379	0x20	In	IN(12)	Generic Input
13	0x379	0x10	In	IN(13)	Generic Input
15	0x379	0x08	In	IN(15)	Generic Input
18-25	-	-	Gnd	-	Ground

RJ-45 CONNECTORS

INPUTS - IN(10-15)

The 5 generic inputs are real time inputs that can be read by the computer. The OPCS software can be configured to make use of these inputs by specifying the corresponding port/mask via the OPCSDEFS.OPC or HOMEDEFS.HOM files.

Typically generic inputs are used for either home sensors or buckle/viewer switch sensing.

IN(10) - IN(15)
RJ-45 PINOUTS

PIN#	DESCRIPTION	COLOR
1	Chassis	WHT/ORN
2	GND	ORN
3	Chassis	WHT/GRN
4	-	BLU
5	Chassis	WHT/BLU
6	IN	GRN
7	Chassis	WHT/BRN
8	+12	BRN

OUTPUTS - OUT(2-7)

The 6 generic outputs can be controlled directly by commands in HOMEDEFS.HOM or OPCSDEFS.OPC, e.g. the SETBIT, CLRBIT, and XORBIT commands.

Typically, generic outputs are used for deenergizing motors to allow manual load/unload of film with the custom LOAD and LINEUP commands.

OUT(2) - OUT(7)
RJ-45 PINOUTS

PIN#	DESCRIPTION	COLOR
1	Chassis	WHT/ORN
2	GND	ORN
3	Chassis	WHT/GRN
4	-	BLU
5	Chassis	WHT/BLU
6	OUT	GRN
7	Chassis	WHT/BRN
8	+12	BRN

TENSION OUTPUTS - TEN(8) AND TEN(9)

The tension motor outputs TEN(8) and TEN(9) can control the FEED and TAKEUP motors for camera and projector.

When parallel port pin 8's bit changes from 0 to 1, the TEN(8) RJ-45 connector's FEED and TAKEUP outputs will change state, always being the compliment of each other (ie. if FEED is 'on', TAKEUP will be 'off').

### TEN(8) AND TEN(9) ###			
### RJ-45 PINOUTS ###			
PIN#	DESCRIPTION	COLOR	CRYDOM PIN#
1	Chassis	WHT/ORN	4
2	-TAKEUP	ORN	-
3	Chassis	WHT/GRN	3
4	+TAKEUP	BLU	-
5	Chassis	WHT/BLU	4
6	-FEED	GRN	-
7	Chassis	WHT/BRN	3
8	+FEED	BRN	-

POWER OUTPUTS - PWR(1) THRU PWR(3)

PWR-1 through PWR-3 can be used to supply +12V power to the printer.

### PWR-1 THRU PWR-3 ###			
### RJ-45 PINOUTS ###			
PIN#	DESCRIPTION	COLOR	CRYDOM PIN#
1	Chassis	WHT/ORN	4
2	GND	ORN	-
3	Chassis	WHT/GRN	3
4	-	BLU	-
5	Chassis	WHT/BLU	4
6	-	GRN	-
7	Chassis	WHT/BRN	3
8	+12	BRN	-

AUTHOR

Greg Ercolano / Seriss Corporation 2009, 2021

RTMC16(DOCS) Optical Printer Control System RTMC16(DOCS)

NAME

rtmc16 - notes on the Kuper Controls RTMC16 stepper motor control card

DESCRIPTION

The Kuper Controls RTMC16 card is a "long slot" ISA card for the IBM PC that can generate steps/direction pulse streams to control up to 16 stepper motors at once.

This is the first generation Kuper Controls card using mostly simple digital TTL circuitry that involves no firmware. Subsequently it's a a very dense board, but all the parts are user changeable with off the shelf chips.

Bill Tondreau of Kuper Controls supplied these cards. They're somewhat of a rare item though, as they were superseded by the RTMC48.

RTMC16 DOS DRIVER

OPCS communicates with the board via the MS-DOS device driver "MDRIVE.COM", which provides a standard low level interface to the card that OPCS can make use of to run the motors efficiently.

NOTE: The MDRIVE.COM driver must be loaded before running the OPCS software. and can be installed either by the AUTOEXEC.BAT, or by a separate batch script.

If the RTMC16 card's jumpers are default (BaseAddr=300 and IRQ=5), then you can install the driver by just running:

mdrive

If the RTMC16 is configured for a different base address or IRQ, then see below.

CONFIGURING THE BASEADDR AND IRQ

In OPCS K1.xx, the base address is configured in STARTUP.DEFS using the 'baseaddr' command in that file. IRQ not configurable.

In OPCS K2.00 through K2.09, the base address is configured in OPCSDEFS.OPC with the 'baseaddr' command. IRQ not configurable.

In OPCS K2.10 and up, the MDRIVE.COM driver allows both the base address and IRQ to be configured on the command line. The default would be:

```
    mdrive -b300 -i5          <-- Sets base address=0300h, IRQ=5
             |    |
             |    IRQ=5
            Base Addr=300
```

..and if your RTMC16 jumpers are set differently, then specify matching values accordingly. e.g. if the jumpers are set to BaseAddr=340 and IRQ=6, then start the driver with:

```
    mdrive -b340 -i6
```

To list the MDRIVE driver's options, run '**mdrive -help**'. If it does not show a list of options, then it is an older version that does not support command line options.

TECHNICAL SYNOPSIS

When the software wants to move a motor, it provides 16 separate 12 bit velocity values (one per motor channel) that we can call a 16 channel 'sample', and 120 of these samples are sent per second to the card using the RTMC16's hardware interrupt on IRQ 5.

The software has to keep up with this transmission rate, otherwise it will lose track of the motor positions.

The MDRIVE.COM device driver provides a 64k motor velocity circular ring buffer that the OPCS software constantly writes to while running the motors. The OPCS software uses INT 99h to communicate with the driver, providing the address of the ring buffer, and start/stop commands.

The RTMC16 card generates 120 interrupts per second to the MDRIVE.COM driver, which feeds out 16 velocities per interrupt from the ring buffer to the card, rotated out to the motors on the NEXT interrupt tick.

LOGIC CONNECTOR

For the RTMC16 board, it is generally not recommended to use the built in "Logic Connector" for anything but inputs. It's best to use the parallel ports, or add-on 8255 based GPIO boards if lots of digital I/O is needed.

KUPER CONTROL RTMC16 CARD CONNECTOR 'P2'
(DB37S - 37 pin connector)

```
--------------------------------
   1  - (*)    20 - +5VDC
   2  - STEP A 21 - DIR A
   3  - STEP B 22 - DIR B
   4  - STEP C 23 - DIR C
   5  - STEP D 24 - DIR D
   6  - STEP E 25 - DIR E
   7  - STEP F 26 - DIR F
   8  - STEP G 27 - DIR G
   9  - STEP H 28 - DIR H
   10 - STEP I 29 - DIR I
   11 - STEP J 30 - DIR J
   12 - STEP K 31 - DIR K
   13 - STEP L 32 - DIR L
   14 - STEP M 33 - DIR M
   15 - STEP N 34 - DIR N
   16 - STEP O 35 - DIR O
   17 - STEP P 36 - DIR P
   18 - (*)    37 - +5VDC
   19 - (*)
```

DB37S (37 pin connector)

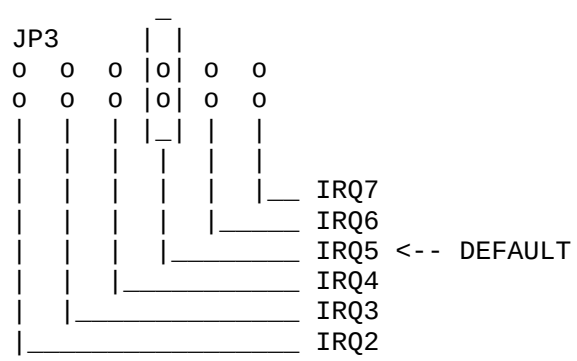
(*) = Jumper Select GND or +5 with JP5:
+5VDC - Short pins 1 & 2 on JP5
GND - Short pins 2 & 3 on JP

NOTE: All outputs are OPEN COLLECTOR TTL. Maximum +5 current draw should not exceed 400 milliamps.

KUPER CONTROL CARD IRQ JUMPER SETTINGS (JP3)
Selects the IRQ used for feeding velocities

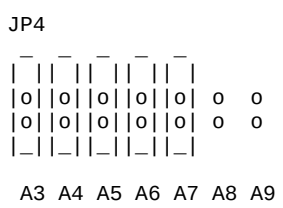
```
--------------------------------
```

JP3 has a single jumper on one set of pins to set the IRQ;
from left to right, pins set IRQ 2 thru 7, with 5 being the default:



KUPER CONTROL CARD SWITCH SETTINGS 'JP4'
Selects the KuperBase address value
(Jumpers A3-A9)

This is the default configuration for 0300h:



Here's the table of the JP4 jumper variations from the RTMC16 manual, shown in "most likely to work" order:

KuperBase Address	A3	A4	A5	A6	A7	A8	A9
0300	1	1	1	1	1	0	0
0320	1	1	0	1	1	0	0
0320	0	1	0	1	1	0	0
0330	1	0	0	1	1	0	0
0340	1	1	1	0	1	0	0
0280	1	1	1	1	0	1	0
02a0	1	1	0	1	0	1	0
0308	0	1	1	1	1	0	0
0310	1	0	1	1	1	0	0
0318	0	0	1	1	1	0	0

NOTE: 0 = off 1 = on

Factory setting is 0300, and there is usually no need to modify this setting unless other boards in the machine are conflicting with this address. Same for the IRQ setting.

KUPER LOGIC CONNECTOR [R1]
Inputs are tied high to +5

PIN	NAME	PORT	MASK (hex)
1	GND	GND	GND
2	out 0	0x306	01
3	out 1	0x306	02
4	out 2	0x306	04
5	out 3	0x306	08
6	out 4	0x306	10
7	out 5	0x306	20
8	out 6	0x306	40
9	out 7	0x306	80
10-12	???	???	??
13	+5	+5	+5
14	GND	GND	GND
15	in 0	0x306	01
16	in 1	0x306	02
17	in 2	0x306	04
18	in 3	0x306	08
19	in 4	0x306	10
20	in 5	0x306	20
21	in 6	0x306	40
22	in 7	0x306	80
23-24	???	???	??
25	+5	+5	+5

KUPER "INDUSTRIAL" CARD

The Kuper Controls "Industrial Card" is a 'half slot ISA' card, a variation on the RTMC-48. So for OPCS, install the "RTMC48.COM" driver.

The "H1" 40 pin connector (upper-left) is the steps/direction. The "H2" 40 pin connector (upper-right) is the "logic" connector. For OPCS, only the "H1" connector should be used.

On the H1 connector, pin #1 is at the lower-left of the connector (component side facing you).

This card has 3 jumper blocks, whose "factory" settings are:

```
   JP1:  o-o o            -- sets voltage for pin #1 (OPCS: don't care)

   JP2:  o o o o o        -- sets samples-per-second(?) (default: 120/sec)
         | | |
         o o o o o

   JP3:  o o o o o o      -- sets the IRQ (default IRQ 5)
               |
         o o o o o o
```

..where '-' is a horizontal jumper, and '|' is a vertical jumper.

KUPER PORT MONITOR PROGRAM

The OPCS software comes with kuper.exe, a program that monitors the real time status of the Kuper logic port. Run 'kuper.exe'. This tool can be downloaded from http://seriss.com/opcs/ftp/

SEE ALSO

RTMC48(DOCS) - notes on the Kuper Controls RTMC48 motor control card
A800(DOCS) - notes on the OPCS/Seriss Corporation A800 motor control card
8255(DOCS) - how to control 8255 based digital I/O cards
KUPER(DOCS) - documentation on the kuper card connectors
SD-1600(DOCS) - 16 channel stepper distribution board (simplify DB-37 wiring)

ORIGIN

Gregory Ercolano, Topanga, California 04/12/00

NAME

rtmc48 - notes on the Kuper Controls RTMC48 stepper motor control card

DESCRIPTION

The Kuper Controls RTMC48 card is a "long slot" ISA card for the IBM PC that can generate steps/direction pulse streams to control up to 48 stepper motors at once.

Bill Tondreau of Kuper Controls supplies these cards, and OPCS can it to control up to 16 motor channels with it.

RTMC48 DOS DRIVER

OPCS communicates with the board via the MS-DOS device driver "RTMC48.COM", which provides a standard low level interface to the card that OPCS can make use of to run the motors efficiently.

NOTE: The RTMC48.COM driver must be loaded before running the OPCS software. and can be installed either by the AUTOEXEC.BAT, or by a separate batch script.

If the RTMC card's jumpers are default (BaseAddr=300 and IRQ=5), then you can install the driver by just running:

rtmc48

CONFIGURING THE BASEADDR AND IRQ

In OPCS K1.xx, the base address is configured in STARTUP.DEFS using the 'baseaddr' command in that file. IRQ not configurable.

In OPCS K2.00 through K2.09, the base address is configured in OPCSDEFS.OPC with the 'baseaddr' command. IRQ not configurable.

In OPCS K2.10 and up, the RTMC48.COM driver allows both the base address and IRQ to be configured on the command line. The default would be:

```
rtmc48 -b300 -i5          <-- Sets base address=0300h, IRQ=5
         |    |
         |    IRQ=5
         Base Addr=300
```

..and if your RTMC48 jumpers are set differently, then specify matching values accordingly. e.g. if your RTMC48 card has jumpers set to BaseAddr=340 and IRQ=6, then start the driver with:

rtmc48 -b340 -i6

To list the RTMC48 driver's options, run 'rtmc48 -help'. If it does not show a list of options, then it is an older version that does not support command line options.

TECHNICAL SYNOPSIS

When the software wants to move a motor, it provides 48 separate 12 bit velocity values (one per motor channel) that we can call a 48 channel 'sample', and 120 of these samples are sent per second to the card using the RTMC48's hardware interrupt on IRQ 5.

The software has to keep up with this transmission rate, otherwise it will lose track of the motor positions.

The RTMC48.COM device driver provides a 64k motor velocity circular ring buffer that the OPCS software constantly writes to while running the motors. The OPCS software uses INT 99h to communicate with the driver, providing the address of the ring buffer, and start/stop commands.

The RTMC48 card generates 120 interrupts per second to the RTMC48.COM driver, which feeds out 48 velocities per interrupt from the ring buffer to the card, rotated out to the motors on the NEXT interrupt tick.

LOGIC CONNECTOR

Controlling the Kuper Logic connector changed with the new RTMC48. Assuming the kuper base port is 0300:

```
#define BASE 0x300

/* WRITE 8 BITS TO THE LOGIC PORT */
out(BASE+0x14h, byte);

/* READ 8 BITS FROM THE LOGIC PORT */
out(BASE+0x15h, 0x40);          /* disable chip select */
byte = inp(BASE);               /* read data from BASE */
```

Note that it is now easy to use the output port (it's dedicated), but is hard to do a read (it involves two operations).

For using the OPCS software with the RTMC48, it is NOT RECOMMENDED you use the Kuper Logic connector for reading data. However, writing data appears to be ok.

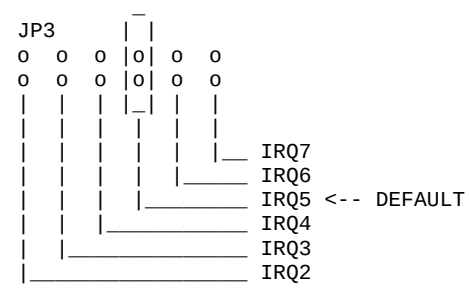
This is in contrary to the recommendations for using OPCS with the RTMC16, where use of the logic connector can be used for inputs only.

KUPER CONTROL RTMC16 CARD CONNECTOR 'P2'
(DB37S - 37 pin connector)

1 - (*)	20 - +5VDC	
2 - STEP A	21 - DIR A	DB37S (37 pin connector)
3 - STEP B	22 - DIR B	
4 - STEP C	23 - DIR C	
5 - STEP D	24 - DIR D	
6 - STEP E	25 - DIR E	
7 - STEP F	26 - DIR F	(*) = Jumper Select GND or +5 with JP5:
8 - STEP G	27 - DIR G	+5VDC - Short pins 1 & 2 on JP5
9 - STEP H	28 - DIR H	GND - Short pins 2 & 3 on JP
10 - STEP I	29 - DIR I	
11 - STEP J	30 - DIR J	
12 - STEP K	31 - DIR K	NOTE: All outputs are OPEN COLLECTOR
13 - STEP L	32 - DIR L	TTL. Maximum +5 current draw
14 - STEP M	33 - DIR M	should not exceed 400 milliamps.
15 - STEP N	34 - DIR N	
16 - STEP O	35 - DIR O	
17 - STEP P	36 - DIR P	
18 - (*)	37 - +5VDC	
19 - (*)		

KUPER CONTROL CARD IRQ JUMPER SETTINGS (JP3)
Selects the IRQ used for feeding velocities

JP3 has a single jumper on one set of pins to set the IRQ;
from left to right, pins set IRQ 2 thru 7, with 5 being the default:



KUPER CONTROL CARD SWITCH SETTINGS 'JP4'
Selects the KuperBase address value
(Jumpers A3-A9)

This is the default configuration for 0300h:

Here's the table of the JP4 jumper variations from the
RTMC16 manual, shown in "most likely to work" order:

KuperBase Address	A3	A4	A5	A6	A7	A8	A9
0300	1	1	1	1	1	0	0
0320	1	1	0	1	1	0	0
0320	0	1	0	1	1	0	0
0330	1	0	0	1	1	0	0
0340	1	1	1	0	1	0	0
0280	1	1	1	1	0	1	0
02a0	1	1	0	1	0	1	0
0308	0	1	1	1	1	0	0
0310	1	0	1	1	1	0	0
0318	0	0	1	1	1	0	0

NOTE: 0 = off 1 = on

Factory setting is 0300, and there is usually no need
to modify this setting unless other boards in the machine
are conflicting with this address. Same for the IRQ setting.

KUPER LOGIC CONNECTOR [R1]
Inputs are tied high to +5

PIN	NAME	PORT	MASK (hex)
1	GND	GND	GND
2	out 0	0x306	01
3	out 1	0x306	02
4	out 2	0x306	04
5	out 3	0x306	08
6	out 4	0x306	10
7	out 5	0x306	20
8	out 6	0x306	40
9	out 7	0x306	80
10-12	???	???	??
13	+5	+5	+5
14	GND	GND	GND
15	in 0	0x306	01
16	in 1	0x306	02
17	in 2	0x306	04
18	in 3	0x306	08
19	in 4	0x306	10
20	in 5	0x306	20
21	in 6	0x306	40
22	in 7	0x306	80
23-24	???	???	??
25	+5	+5	+5

KUPER "INDUSTRIAL" CARD

The Kuper Controls "Industrial Card" is a 'half slot ISA' card, a variation on the RTMC-48. So for OPCS, install the "RTMC48.COM" driver.

The "H1" 40 pin connector (upper-left) is the steps/direction. The "H2" 40 pin connector (upper-right) is the "logic" connector. For OPCS, only the "H1" connector should be used.

On the H1 connector, pin #1 is at the lower-left of the connector (component side facing you).

This card has 3 jumper blocks, whose "factory" settings are:

```
JP1:  o-o o            -- sets voltage for pin #1 (OPCS: don't care)

JP2:  o o o o o        -- sets samples-per-second(?) (default: 120/sec)
      | | |
      o o o o o

JP3:  o o o o o o      -- sets the IRQ (default IRQ 5)
            |
      o o o o o o
```

..where '-' is a horizontal jumper, and '|' is a vertical jumper.

KUPER PORT MONITOR PROGRAM

The OPCS software comes with kuper.exe, a program that monitors the real time status of the Kuper logic port. Run 'kuper.exe'. This tool can be downloaded from http://seriss.com/opcs/ftp/

SEE ALSO

RTMC16(DOCS) - notes on the Kuper Controls RTMC16 motor control card
A800(DOCS) - notes on the OPCS/Seriss Corporation A800 motor control card
8255(DOCS) - how to control 8255 based digital I/O cards
KUPER(DOCS) - documentation on the kuper card connectors

ORIGIN

Gregory Ercolano, Topanga, California 04/12/00

SD-1600(DOCS) Optical Printer Control System SD-1600(DOCS)

NAME

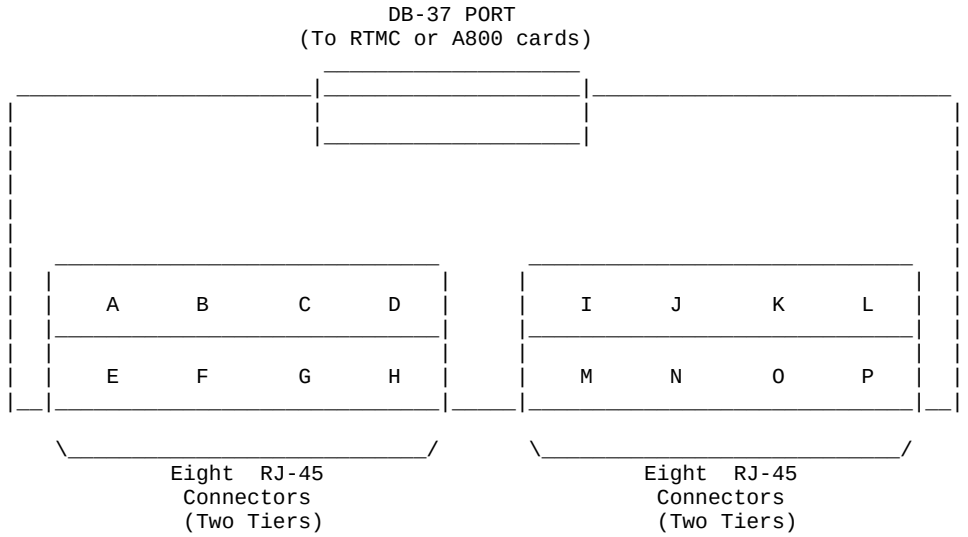
sd-1600 - OPCS 16 channel "stepper distribution" (SD) card

DESCRIPTION

The OPCS "Stepper Distribution" cards (SD-800, SD-1600) were designed to simplify wiring between the computer step pulse generator card (e.g. RTMC16, RTMC48, Kuper Industrial, A800..) and the motor drive modules (Centent, Gecko, LeadShine, etc) by breaking out the DB-37 connector into separate RJ-45 patch cables, one per stepper drive channel.

This board really has no active features on it, other than a fanout to simplify wiring. Optional pullup resistor networks can be used if the application requires open collector outputs from the card to be pulled up to +5V for the idle state to prevent noise.

As of this writing, there is only one version of the board, REV 0, which looks like this:



Typically the female DB-37 connector on the board is connected to the DB-37 connector on the ISA stepper pulse generator card plugged into the the DOS computer using 6' male/male DB-37 cable.

And separate RJ-45 patch cables are wired to the A/B/C/D..M/N/O/P ports at the bottom of the board, which run out to the individual stepper motor driver modules.

The DB-37 follows Kuper's pinout; see 'man kuper' for more info.

The RJ-45 pinout diagram is documented on the board's silk screen, but is basically:

```
          RJ-45              WIRE      CENTENT       GECKO        LEADSHINE
          PIN#  SIGNAL       COLOR (*) DRIVE         DRIVE        DRIVE
          ----  -----------  --------  ------------  -----------  ------------
           1    CHASSIS GND  -         N/C           N/C          N/C
           2    CHASSIS GND  -         N/C           N/C          N/C
         _ 3    CHASSIS GND  -         N/C           N/C          N/C
   DIR |   4    DIRECTION    BLU       DIRECTION     (8) DIR      DIR-(DIR)
       |_  5    +5V          WHT/BLU   +5 VOLTS DC   (10) COMMON  DIR+(5V-24V)
         _ 6    CHASSIS GND  -         N/C           N/C          N/C
   STP |   7    +5V          WHT/BRN   N/C           N/C          PUL+(5V-24V)
       |_  8    STEPS        BRN       STEP PULSE    (9) STEP     PUL-(PUL)
```

(*) Premade RJ-45 patch cables for cat5 and cat5e usually have the standard wire colors shown above. For the signals used, the wiring colors are the same for 568A and 568B.

The "chassis ground" signals are derived from the DB-37 metal shell, and the metal shell around the RJ-45 eight-connector-blocks. The signal path for the chassis ground is not required, and indeed the DB-37 female and RJ-45 connectors don't always have a metal shell provided.

The RJ-45 cables are typically long cat5 or cat6 patch cables that are cut in half; the cut ends stripped and tinned with solder to connect the needed 4 conductors to the corresponding screw terminals on the motor drives.

The Gecko and Centent drives only need three conductors:

1. Steps (BRN)
2. Direction (BLU)
3. A single +5V wire common for the above two signals (WHT/BLU or WHT/BRN)

The newer motor drives from China all have separate +5V connections for each of the signals, so all 4 conductors are used:

> Steps (BRN)
> +5V for steps (WHT/BRN)
> Direction (BLU)
> +5V for direction (WHT/BLU)

Please note these signals are DIRECTLY FROM THE COMPUTER MOTHERBOARD, so be very careful with them. Do not let them short to chassis ground or to each other.

RJ-45 CABLE TERMINATION

See OPCSIFACE(DOCS) for info on RJ-45 cable termination complete with diagrams and info about cable preparation, strain relief, etc.

AUTHOR

Greg Ercolano / Seriss Corporation 2023

SD-800(DOCS) Optical Printer Control System SD-800(DOCS)

NAME

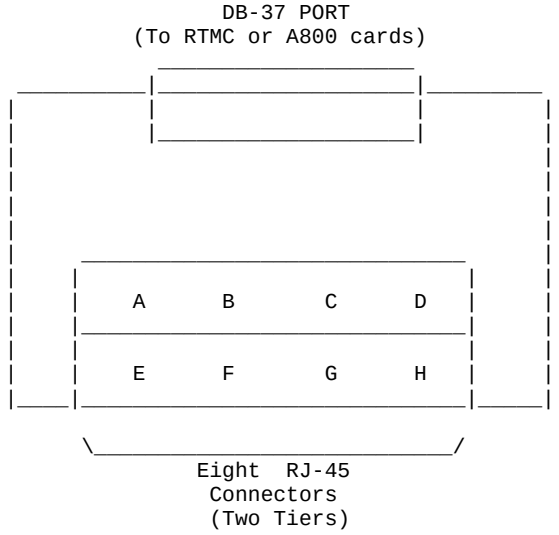
sd-800 - OPCS 8 channel "stepper distribution" (SD) card

DESCRIPTION

The OPCS "Stepper Distribution" card (SD-800) was designed to simplify wiring between the computer step pulse generator card (e.g. RTMC16, RTMC48, Kuper Industrial, A800..) and the stepper motor driver modules (Centent, Gecko, LeadShine, etc) by breaking out the DB-37 connector into separate RJ-45 patch cables, one per stepper drive channel.

This board really has no active features on it, other than a fanout to simplify wiring. Optional pullup resistor networks can be used if the application requires open collector outputs from the card to be pulled up to +5V for the idle state to prevent noise.

As of this writing, there is only one version of the board, REV 0, which looks like this:



Typically the female DB-37 connector on the board is connected to the DB-37 connector on the ISA stepper pulse generator card plugged into the the DOS computer using 6' male/male cable.

And separate RJ-45 patch cables are wired to the A/B/C/D.. ports at the bottom of the board, which run out to the individual stepper drives (Centent, Gecko, LeadShine, etc).

The DB-37 follows Kuper's pinout; see 'man kuper' for more info.
The RJ-45 pinout diagram is on the board, but is basically:

```
          RJ-45                WIRE       CENTENT        GECKO         LEADSHINE
          PIN#  SIGNAL         COLOR (*)  DRIVE          DRIVE         DRIVE
          ----  -----------    ---------  ------------   ------------  ------------
           1    GND            -          N/C            N/C           N/C
           2    GND            -          N/C            N/C           N/C
         _ 3    GND            -          N/C            N/C           N/C
   DIR |   4    DIRECTION      BLU        DIRECTION      (8) DIR       DIR-(DIR)
       |_  5    +5V            WHT/BLU    +5 VOLTS DC    (10) COMMON   DIR+(5V-24V)
         _ 6    GND            -          N/C            N/C           N/C
   STP |   7    +5V            WHT/BRN    N/C            N/C           PUL+(5V-24V)
       |_  8    STEPS          BRN        STEP PULSE     (9) STEP      PUL-(PUL)
```

(*) Premade RJ-45 patch cables for cat5 and cat5e usually have
the standard wire colors shown above. For the signals used,
the wiring colors are the same for 568A and 568B.

Basically only 4 of the 8 wires are used. In some cases only 3 wires are used (Centent & Gecko).

Please note these signals are DIRECTLY FROM THE COMPUTER MOTHERBOARD, so be very careful with them. Do not let them short to chassis ground on the printer, or to each other.

RJ-45 CABLE TERMINATION

See OPCSIFACE(DOCS) for info on RJ-45 cable termination complete with diagrams and info about cable preparation, strain relief, etc.

AUTHOR

Greg Ercolano / Seriss Corporation 2021

```
SERIAL(DOCS)            IBM PC Device Documentation            SERIAL(DOCS)

    NAME
        serial - serial port pinout

    INT 14H   BIOS Serial Communications

        AH=0    INITIALIZE PORT ---------------------------------------------

        AL:
            7       6       5       4       3       2       1       0
            --- baud rate ---       -parity -    stopbit    -- word length --
            000 - 110              x0 - none    0 - 1      10 - 7 bits
            001 - 150              01 - odd     1 - 2      11 - 8 bits
            010 - 300              11 - even
            011 - 600
            100 - 1200
            101 - 2400
            110 - 4800
            111 - 9600
        DX: 0=com1, 1=com2

        AH=1    SEND CHARACTER ----------------------------------------------

        AL: character to send
        DX: 0=com1, 1=com2

        ON RETURN:
            bit 7 of AH set if error occurred

        AH=2    RECEIVE CHARACTER -------------------------------------------


        AL: character read
        DX: 0=com1, 1=com2


    PORT  DLAB=0          DLAB=1
    3f8 - data in/out     baud rate low
    3f9 - IER int enable  baud rate high
```

```
    --- Ports when DLAB is set ----------------------------------------

        03f9 03f8   ACTUAL BAUD RATE
         09   00    50
         04   17    110
         01   80    300
         00   60    1200
         00   30    2400
         00   18    4800
         00   0c    9600
         00   01    real fast

    --- Ports when DLAB is clear ---------------------------------------

        03f8 - 8 bit data i/o (characters in/characters out)

        03f9 Interrupt Enable Register (IER)
            0 - Interrupt when character ready
            1 - Interrupt when transmitter holding register empty (THRE)
            2 - Interrupt when LSR has data ready
            3 - Interrupt when MSR has data ready
            4 - 0
            5 - 0
            6 - 0
            7 - 0

3fa - IIR Interrupt Identification Register

        PortValue  Description (Cause)
        ---------  ------------------------------------------------------
        06h        LSR has info (overrun/parity/framing/break interrupt)
        04h        Data ready (character waiting)
        02h        Transmitter Holding Register Empty (character was sent)
        00h        MSR has info (CTS, DSR, RI, RLSD)


3fb - LCR line ctrl
        0 word length (see below)
        1 word length (see below)
        2 Stop Bits
        3 Parity Enable
        4 Even Parity
        5 Stick Parity
        6 Set Break
        7 DLAB bit (access baud rate via 3f8 and 3f9)

        bit1  bit0  Word Length
        ----  ----  -----------
         0      0   5 bits
         0      1   6 bits
         1      0   7 bits
         1      1   8 bits
```

```
3fc - MCR modem ctrl
      0 - Data Terminal Ready
      1 - Request To Send
      2 - Out 1
      3 - Out 2
      4 - Loop
      5 - 0
      6 - 0
      7 - 0

3fd - LSR line status
      0 - Data Ready
      1 - Overrun Error
      2 - Parity Error
      3 - Framing Error
      4 - Break INterrupt
      5 - Transmitter Holding Reg Empty
      6 - Transmitter Shift Reg Empty
      7 - (always zero)

3fe - MSR modem status
      0 - Delta Clear To Send
      1 - Delta Data Set Ready
      2 - Trailing Edge Ring Detector
      3 - Delta Rx Line Signal Detect
      4 - Clear To Send
      5 - Data Set Ready
      6 - Ring Indicator (=1 during ring voltage)
      7 - Receive Line Signal Detect (=1 when carrier appears)

C:> mode com1 9600,n,8,1
C:> debug
- o 3fb 80
- o 3f8 00
- o 3f9 0c
- o 3fb 03
- o 3fc 03

// UNTESTED
//    Might need to read LSR again to clear Data Ready bit?
//
while ( 1 )
    if ( inp(0x03fd) & 0x01 )
        fprintf(stderr, "%c", inp(0x03f8));
```

MODEM/SERIAL PORT WIRING

25 PIN DCE (IBM computer, modems)			25 PIN DTE (terminal)		
1	(shield)		1		
2	TX ----	OUT	2	RX	IN
3	RX	IN	3	TX	OUT
4	RTS ---	OUT	4		
5	CTS	IN	5		
6	DSR	IN	6		
7	GND	GND	7		
8	CD/RLSD	IN	8		
9	(tx currentloop)		9		
10			10		
11			11		
12			12		
13			13		
14			14		
15			15		
16			16		
17			17		
18	(rx currentloop)		18		
19			19		
20	DTR ---	OUT	20		
21			21		
22	RI	IN	22		
23			23		
24			24		
25	(rx currentloop ret)		25		

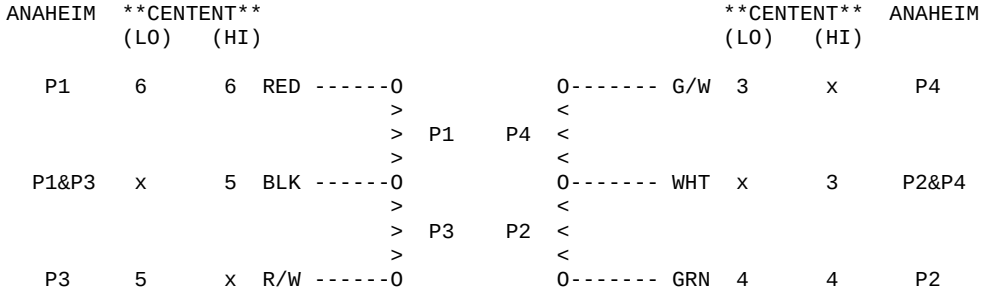
9 PIN DCE (IBM computer, modems)			9 PIN DTE (terminal)		
1	CD/RLSD	IN	1		
2	TX ----------	OUT	2	RX	IN
3	RX	IN	3	TX ----	OUT
4	DTR ----------	OUT	4		
5	GND	GND	5		
6	DSR	IN	6		
7	RTS ----------	OUT	7		
8	CTS	IN	8		
9	RI	IN	9		

NAME

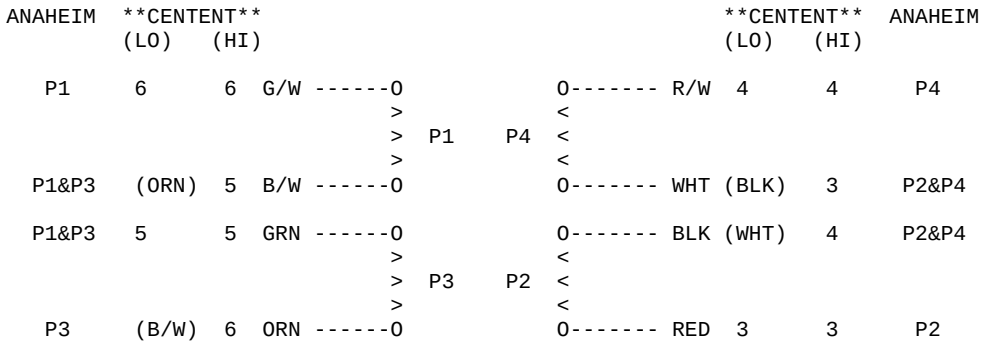
slosyn - slosyn motor wiring notes

SUPERIOR ELECTRIC 6 WIRE SLO-SYN
INTERNAL WIRING DIAGRAM
Centent and Anaheim terminals shown

To rotate the motor, energize coils sequentially in the following order: P1, P2, P3, P4... I know it looks wrong, but that's how it works.



SUPERIOR ELECTRIC 8 WIRE SLO-SYN
INTERNAL WIRING DIAGRAM
Centent and Anaheim terminals shown



SYNTAX(DOCS) Optical Printer Control System SYNTAX(DOCS)

NAME

syntax - syntax of OPCS numeric and math expressions

OPCS COMMAND SYNTAX

Many commands can appear on one line, and a line can be up to 128 characters (or can wrap around the screen about one and a half times). Commands and arguments are separated by white space (spaces or tabs).

Commands that allow a variable number of fixed arguments, e.g. SEEK, RES, and RAT, they let you use '-' to skip arguments.

Where appropriate, '>' can be used as a prefix to a numeric value to specify an absolute position in the context of frames to shoot, or motor positions to go to.

Some commands expect frame specifications, which can be expressed in many different ways. Such a command is the 'CAM' command, and here are some valid frame specifications:

```
cam 12                  # run camera 12 frames
cam -5                  # run camera 5 frames in reverse
cam 12'2                # run camera 12 feet 2 frames
cam -15'0               # run camera in reverse 15 feet
cam >34                 # send camera TO frame 34
cam >-34                # send camera TO frame -34
cam >-12'3              # send camera TO negative 12 ft 3 frms
cam (3+12*12)           # run camera 147 frames
cam (3+(3*sqrt(16)*12)) # same as above
cam >(3+12*12)          # send camera TO frame 147
```

MATH EXPRESSIONS

You can usually use math expressions in place of most numeric arguments as long as the expression is ENCLOSED IN PARENTHESES, and DOES NOT CONTAIN EMBEDDED SPACES. Example:

```
(3+(3*sqrt(16)*12))
```

Math can be done on frame counter values:

```
(cam+3)
```

For a complete list of all built in math operations, execute:

```
(?)
```

The following lists some of the operations supported by the math expression parser:

```
/*** TYPICAL OPERATIONS ***/

(3+4-2*12/6)       # add, subtract, multiply, divide
(533%256)          # modulus
(2^4)              # exponentiation (powers)

/*** OPCS VALUES ***/
cam   - camera counter value
pro   - main projector counter value
pro1  - main projector counter value
pro2  - aerial projector counter value

/*** MATH FUNCTIONS ***/

sqrt(),  log(),     exp(),
sin(),   cos(),     tan(),
asin(),  acos(),    atan(),
atan2(), radians(), degrees()
hex(), pi

/*** NUMERIC EXPRESSIONS ***/
-12           # negative 12
+34           # positive 34
0x3ff         # hex representation for 1023 decimal
```

DIFFFERENT APPROACHES YIELD SIMILAR RESULTS

With few exceptions, there are usually two ways to do anything in the OPCS software. The following examples show two possible ways to do step printing:

```
do 12 cam 1 pro 2  # take 1 frame of every second projector frame

rat 2 1 rep 12     # same as above
```

The following two examples show how to automate the wedging process:

```
do 200 cam 2 pse   # shoots 2 frames with a keyboard pause inbetween

cam 2              # hitting the F4 key each time will shoot 2 frames
                   # doing more or less the same as the above
```

COMMAND LINE EDITING KEYS

OPCS supports all of DOS's editing keys, such as F3 (retype a line but do not execute), LEFT and RIGHT ARROW keys for revealing characters from the last line typed, INSERT and DELETE keys, etc. Refer to your DOS manual for more on these editing keys and their use.

In addition to the standard DOS editing keys, the F4 key is programmed by the OPCS software to RE-EXECUTE the last line typed with one key press. This is useful for doing wedges, or situations where manual stopping/starting is necessary.

SHELL EXECUTION

Part of the system philosophy is to allow the user to their own custom programs/scripts from within OPCS, just like DOS commands. You can execute DOS commands by prefixing the command with a 'bang' (!), e.g.

```
!  dir *.run  |  more
```

You can also use (!) to QUOTE a dos command, so that OPCS commands can be mixed with the dos commands:

```
cam 12 ! echo Shot 12 frames !  pro 12 ! echo DONE
         ------------------            ---------
```

The underlined commands are executed as DOS commands, the rest are executed as OPCS commands. Note how '!' turns DOS execution on and off throughout the line.

This (!) technique is patterned after a similar technique used by programs that run under the UNIX operating system.

ORIGIN

Gregory Ercolano, Los Feliz California 12/15/89

VERSIONS(DOCS) Optical Printer Control System VERSIONS(DOCS)

OPCS VERSION HISTORY/RELEASE NOTES

These are in order of most recent versions first, oldest last.

K2.21/TC - 06/04/2024

> Added OPCS_NOMOTOR_FRAME_DELAY environment variable to control
the per-frame delay when simulating shutter runs with 'motors off'.
Default is zero (no delay).

> Restored Allstop key during 'motors off' shutter simulation,
which was being ignored. (Needed to DisableKeyboardInts()
for new allstop keyboard checking to work correctly)

> Applied proofreader mods to manual pages. (MM)

> Replaced break_up_line() with better, regression tested
code that passes on linux with valgrind.

K2.20/TC - 05/11/2024

> a800drv.com MUST block kb ints during motor runs.
We /have/ to. If user keeps typing while motors are running
with kb ints on, the buffer will fill and starts BEEPING
which causes missed feeds to motors causing overtravel,
and WITHOUT tripping a sync fault..!

> Fixed weird issue with how INT 10 scrollup intercept handled.
Noticed if A800DRV was installed, in VI if I G2 to goto line 2
in a file, then hit 'dd', the line above would repeat into
line 2. Fixed how comparisons work in that code, and BIOS fallthrus.

> Makefile now maintains VERSION for all programs.
Changed all .C and .ASM code to use new datetime.h/datetime.inc
and made those files a dependency. Tweaked the datetime build tool
to take -asm/-c arguments as well as the VERSION# info.

> Added call to ForceUpdateCounters() to name_defs() so counters
get renamed right away. (Old names A/B/C remained during 'home'
operation.. only fixed up once OPCS fully initialized)

> Moved all keyboard reading stuff to readkey.c::ReadKeyboard()
which now checks the keyboard busy bit before reading inp(0x60),
and handles extended keycodes and such.

Doing this means ALL keyboard reading must use ReadKeyboard(),
and never read inp(0x60) directly, otherwise it will miss key release
events and such. Also, kb interrupts MUST BE OFF for ReadKeyboard()
to work reliably, and the BIOS keyboard buffer must be flushed before
and after, to avoid weirdness occurs.

ReadKeyboard() should probably check (inp(0x21) & 2) to see if
keyboard interrupts are on or off: if on, use kbhit() and getch(),
and if off, read the raw hardware.

K2.15/TC - 05/11/2024 (not released, cleanup)

TODO: Add code to a800drv INT 99 to halt on any fun call
that's out of range. Use "jae" to test the value,
and use existing print hex to show the bad AH value.
Then port this change to RTMC48 and MDRIVE!

o OPCS:
> Supports 80x25, 80x40, 80x50 size screens.
> Removed "Scroll Lock" key stuff, now adjust scroll height directly.

o A800DRV:
> Fixed screen scrolling intercept to support different sized screens.

> Fixed INT 99/AH=05h "set video address".
Was saving AX as address, but AH is already used for func#.
Switched to use BX. (Apparently function unused, so bug unnoticed)

> Unused scroll lock key stuff removed

> Added GetScrollHeight() function (INT 99/AH=13) to return
the current scroll height last set. Used by OPCS to restore
scroll height during DOS commands.

```
       > Unused keyboard intercept stuff removed
         What was that all about anyway? Thought it might be affecting
         key release stuff. It seems unused??
         TODO: PORT THIS CHANGE (AND OTHERS) TO MDRIVE AND RTMC48

         TODO2: Also, it's important that port 21h is saved during
                motor start, and restored on motor stop. The keyboard
                int bit must NOT DISTURBED OR CHANGED THROUGHOUT:

                 If 'key' or 'jog' turns kb ints off, they must REMAIN OFF
                 or the keyboard polling thing won't work; the BIOS KB INT
                 routines will read port 60h, removing the scancode from the
                 hardware buffer such that OPCS's ReadKeyboard() won't see it.
                 This causes it to miss key release events.

    o Key events getting eaten: Is it the printf() library??
      Try not using printf/fprintf to put msgs on screen while
      kb ints are disabled.

    o home.exe (2.21):
        > Fixed sync fault issue caused by kbhit() in krun.c allstop checks
        > Added generation of "fault.txt" info file on sync fault errors.
        > Cleaned up 'home -v' output to be greppable/redirectable
        > Cleaned up print_cmd a bit

    o Rewrote keyboard reading from simple inp(0x60) to smarter
      handling of extended and typematic key codes (e.g. repeating HOME key
      problem)

    o jog: Added extra 4 channels MNOP for full 16 channel support (ferriter)
      UNTESTED WITH ACTUAL HARDWARE! Use 16chan board to test

    o Fixed weird keyboard hang when 'key' mode invokes DOS commands, e.g.
      keyfunc -add "! echo OK" 0060 ff 52  0060 80 80
                  ^^^^^^^^^^^      ^^^^^
                  DOS via OPCS     Insert key (extended)
      Seems on return from execute_string() it's important to call
      clear_kb_buffer(), which uses kbit()/getch() to read any pending chars.
      Doing this before disabling keyboard ints seems to solve it.

     o 'hardware off' - now shutters run 1x at a time at 0.25sec spd
       (cam/pro/etc). This way 'key' mode for pro/cam slew works, and
       doesn't just immediately jump to the huge numbers it uses for slew.

     o readline:
       > Various small tweaks to support different sized screens.
         Also can now make use of last row. (Could not before)

K2.14/TC - 04/23/2024

     o Small code cleanup of opcs.c:
        > Move "Defs" struct initialization to opcsdefs.c -> InitDefs()
        > Keep init_globals() in opcs, which calls above
        > Remove old comments from opcs.c and new InitDefs() function
        > Removed commented out code

     o Small code cleanup of opcskuper.c:
        > Removed commented out code

     o Added ppr to homedefs.hom to affect motor speeds.

K2.13/TC - 04/21/2024

     o opcs.exe and home.exe: fixed bug detecting motor resident routines.
       They both would test a byte of the segement of the INT 99 vector
       for zero to see if motor routines not installed. But if DOS is
       efficiently stored in high memory, the segment byte CAN be zero,
       causing OPCS to not think motor drivers are installed when they are.
       We now test BOTH bytes of segment register for zero.

     o Fixed a bug that caused intermittant hangs when motors stopped running.
       This was in the A800DRV.COM driver. Particularly noticeable when
       running the HOME program repeatedly, e.g. do 200 home a b c.
       Details:
           Motor driver restores the old IRQ vector when motors come to a stop,
           but the IRQ interrupt service was using DOS (INT 21H) to do this.
           DOS must not be called from WITHIN an interrupt service, even for
           something as simple as changing interrupt vectors. Changed the code
           to change the vector directly, and that solved the instability.

     o gr: only rewrites file if changes were made

     o Created movop - general mocon file operations

     o velrep checked: pretty sure it detects buckle,
```

```
       even tho the manpage for velrep says:

            ---
            FUTURE
                o Add a way to specify BUCKLE/VIEWER/TRIP checks in .vrp file.
                  (We only have AllStop checks currently)
            ---

       That warning does not seem to be true, if it ever was,
       so it was removed from the man pages.

     o readline(): uses BIOS to update cursor and print crlf
       on hitting ENTER, so "opcs > file" still lets user see
       what they're editing.

     o Fixed MRP manpage that was missing the [chan] spec

     o Fixes were made to the 'man ansi' that had errors in describing
       how to program keys using ANSI codes.

     o Fixed SAMPSPERSEC manpage to include new A800 REV-B firmware
       that runs the IRQs at 120 Hz, same value as the RTMC cards.
       Some other man pages were updated for this.

     o Found old calc program's source code, and added it to build,
       added readline() features for editing.

K2.12/TC - 08/19/2023

     o 'show' man page improved to show example output

     o gr: fixed text rendering problem

       Turned out the BEETLE computer's BIOS graphics pixel plot
       (INT 10H) corrupts the app! Wrote my own pixel plotting into
       screen memory and the problem went away. Phoenix BIOS pixel plot
       can't be trusted!

       After writing my own text rendering code in grfont/*, it's probably
       all for naught, and can fall back to printf() and friends, and just
       replace the Plot() code in gr.c with grfont/grfont.c's gr_plot() code.

K2.11/TC - 03/22/2022

     o jog: fixed problem with slew (9) and (3) keys
            "jumping" on short runs.

     o a800drv: Had to detect display: color (b800) vs. mono (b000)
       for beetle to keep up. update_counters updating both color+mono
       seemed to prevent motor updates from being fast enough (?)

       OLD: update_counters was slapping to BOTH b800h and b000h
       NEW: Code now determines video mem address using INT 11H

       XXX: ***WHY*** wasn't the above problem causing a sync fault?!
            If user program not fast enough, should have triggered a fault.
            Look into what was happening/why this is.

K2.10/TC - 02/16/2022

     o Added new "velsav" OPCS command

     o A800DRV.COM + RTMC48.COM + MDRIVE.COM
       Fixed recursion error: kb_int handler was calling 'int 99h' to
       redisplay counters which eventually cause a reentry, tripping a
       "@@@ RECURSION" error w/main app! All previous releases have
       this bug, not only K2.00 - K2.05, but even all K100 releases!

       NOTE: In Feb 2022, backported this fix to 2.05 and older versions;
             see "K2.06" for details.

     o TODO: TEST MDRIVE.COM MODS WITH MOTORS!

---

     o A800DRV.COM - added -b/-i/-h command line flags to set irq/baseaddr.
       This is now the only way to set the IRQ/baseaddr.
       Obsoletes the 'baseaddr' command in OPCSDEFS.OPC and HOMEDEFS.HOM

     o 'baseaddr' removed from the Kuper structure from 'OPCS' and 'HOME'
        applications. NOTE: All DOS drivers must be updated:

                A800DRV.COM - Done K2.10/TC - working 01/17/22 TESTED
                RTMC48.COM  - Done K2.10/TC - working 01/19/22 UNTESTED <-- found above recursion error during testing
                MDRIVE.COM  - DONE K2.10/TC - working 01/21/22 UNTESTED
```

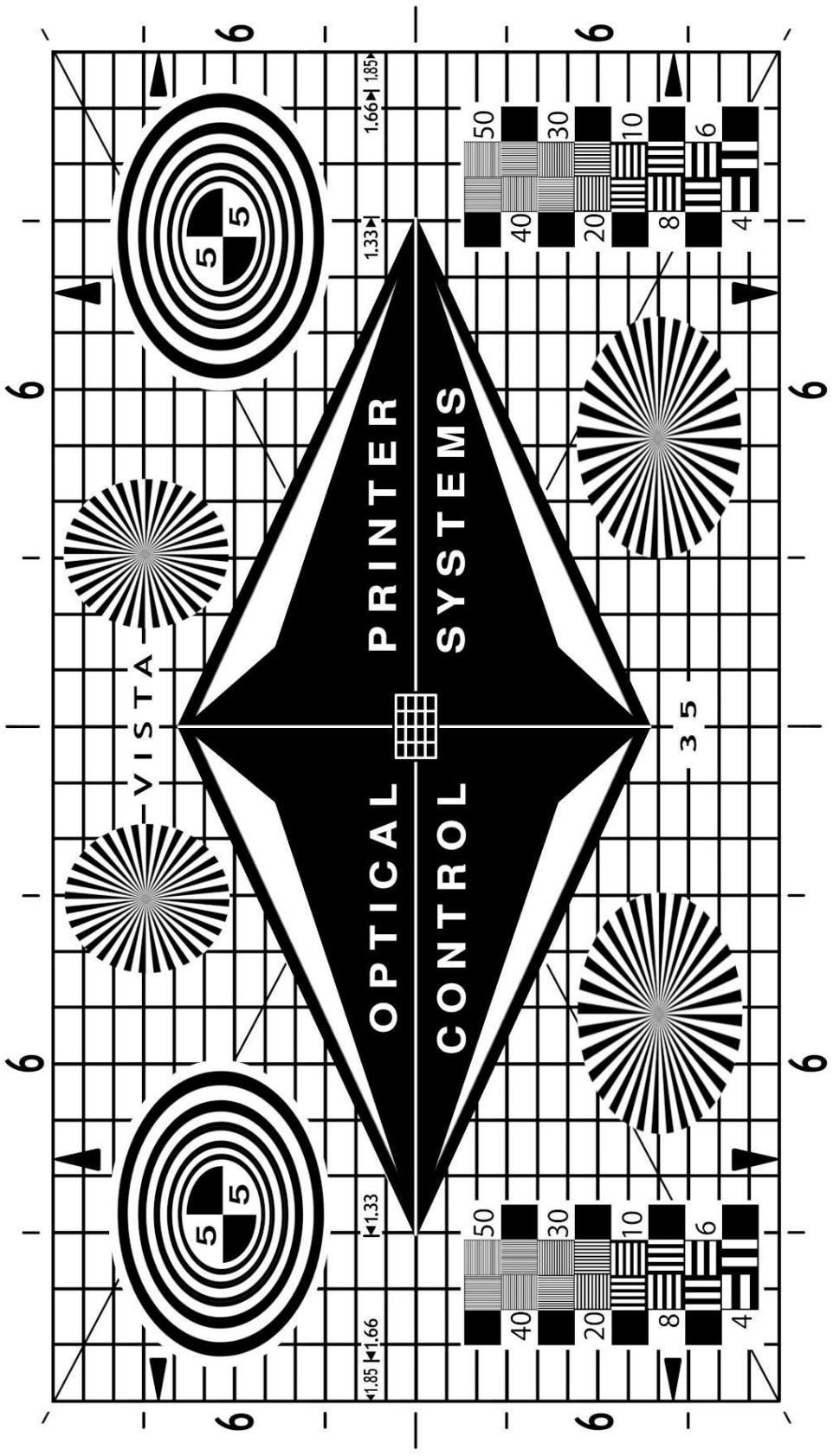
```
     o PrintRing() modified to show DSEG:OFFS for addresses instead of just
       decimal numeric offsets. (Useful for debugging)

     o jog: get rid of "To zero fader use 'cls'" msg (uses cls automatically)

     o jog: 'Reset' now shows blinking cursor at line being edited

     o 'interp d - 0 0 0' (to disable previous interp) is now in man page

     o 'bigcounters yes' now shows warning hashmark on digit overflows

     o Added 'velsav' command

     o (WIP) velrep: add 'lastrep <label>'

K2.06/TC - 02/17/2022
     o THERE'S NO CHANGES TO OPCS IN THIS MAINTEANCE RELEASE,
       but it's being given a number anyway. It's really K2.05
       with a fix to the A800DRV driver.

       Back ported "reentry" recursion fix to A800DRV from K2.10.
       The src fix is in OPCSK200\SRC-BAK\FERRITER.FIX\*.*

       This 2.06 "maintenance release" was *NEEDED*, because in K2.10
       the kuper struct changed size making it ABI incompatible with K2.05.
       (The baseaddr removed from kuper struct in favor of setting it
       as a driver cmd line option).

       So:
           A800DRV.BAD   -- Has reentry BUG (v4.01), released w/K2.05 and older
           A800DRV.ASM   -- Has fix (v4.02), should be given to anyone with
                            K2.00 - K2.05.

K2.05/TC - 07/30/2021

     o 'bigcounters yes': moved cam/pro/rat/fade labels right one char
       to line up with inside of box (like nixie) so single letter labels
       (A,B,C..) indicate CLEARLY which box they're labeling.

K2.04/TC - 07/29/2021  -- Carl Spencer release

     o Changed nixie display to be one line smaller by optimizing
       the use of inverse text. This led to complications due to
       how x,y addressing is affected by the invisible mode chars,
       necessitating a FindXY() function that works but is kinda
       inefficient. See opcsdisp.c for "FUTURE" to improve this maybe.

     o Copied over slap_screen: code from A800DRV.ASM
       to RTMC48.ASM and MDRIVE.ASM (without it, nixie counters
       weren't drawing properly, due to changes in mode chars)

     o Verified runbar shows msg correctly in all counter types

     o Verified counters don't wrap into runbar for all counter types

     o Force jog to always leave cursor in same position so when
       'unlock' is used (U), message prompts during unlock are seen correctly

     o Made sure that running 'rep 5 cam 5 pro 5' shows in runbar
       for all counters: "Rep", then "Cam Shoot", then "Pro Shoot".

K2.03/TC - 03/09/2021

     o Fixed bug in LOG(OPCS) that caused a "division by zero"
       if 'log' was used with FPF(OPCSDEFS) set to 0 for a channel.

     o Fixed bug with: cam >-1'30
       ..where negative footage counts wouldn't correctly range-check
       the trailing frames value

     o Aborted attempt to support floating point fpf.
       Saved work in "floatfpf", previous work in "intfpf".
       See floatfpf.txt for info.

K2.02/TC - 03/07/2021 - Mike Ferriter IMAX <-> 10 Perf 70

     o Fixed bug in 'POST MORTEM'; when debugging enabled,
       junk[] array was [12] instead of [16]

K2.01/TC - 03/04/2021
     o When 'respond' enabled, 'cmdline' forced to 'dos' mode

K2.00/TC - 05/17/2020
     Ported k1.16 to Turbo C 3.0, fixed various bugs in the process.

     o log off: forces log_counters() before closing log
```

```
       (so previous command's effects shown in log before 'log off')

     o flog: fixed bug in fade_log(), where flog=(-1 .. 1)
       was returning the raw fraction (0..1) instead of (start..end).
       SHOULD BACK PORT FIX TO K1.16

     o Got rid of redundant defs->nomotors, replaced with postive logic
       of env->motors.hardware

     o Various fixes to expand_vars/string_replace handling:
           Sometimes args[2] referenced even if NULL
           Variables could expand to garbage if beyond range of args[].
           (Had a > vs >= test error)
           Better warning messages for corner cases

     o Fix commands that act badly when no args given:

        seek check chk go rat rep cam pro pro2 fdi/fdo/dxi/dxo

     o Errors in OPCSDEFS file now report line#.

     o INTERNAL CHANGES:

       > Got rid of all old EMC subroutines, nuked MOTORSUB module.
         Switched to a cleaner module arrangement; see README.txt
         for more info on the module breakdown.

       > RTMC48.ASM modified (now v3.10 - 05/04/2020):
         Changed STI to CLI at top of kuper_int to prevent timer
         interrupt from triggering during motor servicing.
/|\
 |
K200 (Turbo C 3.0 "Pandemic Port" 05/17/2020)

================================================================
================================================================
================================================================

K100
 |
\|/

K1.16 - 05/02/2020
    o Fixed 'show' command not showing if other commands followed it. e.g.
          show cam 1
      ..would not show anything, but ran cam command OK.

    o Added 'check' command, e.g. 'check abc 1000,2000,3000'.
      Also now uses channel names from startup.defs instead of
      hard coded names.

    o Added check for OPCS_NORESSHU variable -- if set,
      using 'go' with abc channels will leave step values in counters.

    o There may be a bug in 'go' with long motor runs, e.g.

        go ab 2000 go ab -8000
                   \_________/
                     This will sometimes run MORE THAN 8000,
                     in a scope grab, it ran 8100 instead of 8000

      Might be a bug in driver's or code's handling when buffer full,
      or handling wrap around. INVESTIGATE! Scope says a800 is sending
      sometimes more/sometimes less pulses than the software sends. WHY?
      Either bug in driver, or bug in 8255 communications timing, or..???

K1.15 - 04/16/08  o velrep would crash if filename didn't exist,
                    or on other file related errors. velrep
                    'Free' routines were not checking for NULL.

                  o velrep docs modified to indicate the + postfix
                    would either increment OR decrement the frame
                    counter based on the velocity's sign (direction).

                  o home: added OPCSDEBUG to help and man page

                  o velrep mem leak: wasn't free()ing stop/goto/loop labels

                  o go mem leak: seems like hfree() wasn't freeing the
                    halloc()s correctly, causing mem to slowly leak.
                    Solution: FreeRingBuffer() hfree() commented out,
                    run hfree() only on opcs_exit()

                  o To make room for more ram:

                    rtmc48/mdrive: Made environment smaller: 25000 -> 20000
```

```
                    (Only need 19660) Run 'defs' or 'show -d' to determine
                    Environment size.

                    Changed MAXSCRIPTS from 20 to 16

                    velrep: Added CHUNKSIZE to realloc()s to prevent
                            mem fragmentation.

                  o Added 'log <MM-DD-YY>" to automatically create
                    a datestamped log file.

K1.14 - 04/27/03  o Adding new OPCS command 'velrep', added VelRep.C.
                    (For Technicolor)
                  o Modified 'man rtmc48' to include kuper diagram.
                  o Fixed bugs in evalnum.c as per linux port
                  o Fixed bug in opcssubs.c as per linux port

K1.13d - 03/15/02 Fixed 'mov' docs; references to 'h' channel changed to 'f'
                  8255.exe modified to let user change outputs (v2)
                  No changes to OPCS that I know of otherwise.

K1.13c - 12/19/00 BUGFIX: seek couldn't handle different PPR values

K1.13b - 02/05/99 BUGFIX: ARGH! All math functions were offset!! ALL WERE WRONG!
                  BUGFIX: seek 10 10 - seek 10 10 seek 10 10 -
                                                 ------------
                                                 This would run at NON-slew spd
                  BUGFIX: (6*(3'0)) and (6+4'0) doesn't work!
                          Unfortunatly all this stuff was broken; parser
                          numeric parser was checking for ' anywhere in
                          the ENTIRE FORMULA, and not stopping at end of
                          numeric value.
                  ENH: Added (?) to print actual evalnum help

K1.13a - 07/11/98 BUGFIX: ease had trouble with numbers >+-32768.
                           Problem was Round() returned an int, and
                           easediff variables were int instead of Pos.
                  ENH: home now has 'reset [chan] [val]'
                  ENH: gr has 'v' to plot vels
                  ENH: pending feeds are now shown in the runbar
                  BUGFIX: BAD BUG in shutter.c since k1.13: 'cam 110' didn't
                          actually shoot 110 frames, and left motor out of
                          sync. Bug was in DumpVels() being called under non-
                          allstop situations.
                  ENH: runcmd now allows -1 as #args for variable args.
                       Also, existence of .HLP files are checked, and printed
                       during errors in number of arguments.
                  ENH: $* now expands ALL variables.

K1.13  - 06/18/98 BUG FIX - 'FDI 2 CAM 10' would stop/go shoot last 8
                  BUG FIX - 'SHU 50 RESET D 0 SEEK 1 1' moves fader?!
                            Created SetCounter() that updates both counter and
                            defs->fader.
                  BUG FIX - mods to shutter.c so key release while in key(OPCS)
                            mode dumps part of ring buffer, making motors stop
                            quicker. [ Added DumpHalfStack() ]
                  ENH: Added cam/pro variables to evalnum
                  ENH: Added 'DO UNTIL (CAM=12) ..', modified manpage
                  DOC: Added manpage for debugger(OPCSDEFS)
                  ENH: home program now has a 'default' clause, so 'home'
                       without args can be _controlled_
                  ENH: Added spdinterp(DEFS) to allow zoom to affect exposure
                       automatically.
                  Introduced: ease.exe and gr.exe for doing pans/zooms.

k1.12f - 05/13/98 Added faderdisplay [on|off] for cinetech

k1.12e - 04/10/98 Added defs:
                       '@', 'echo' 'buckle' 'viewer' 'filter' 'keyfunc'
                  Obsolete:
                       'buckview' 'deenergize' 'keydef'
                  Added opcs: 'autofilt'
                  Seek now ignores viewer, but still senses buckles
                  Redraws allstop msgs w/out flashing after cont/abt.
                  Added autofilt(OPCS) and filter(OPCSDEFS)
                  Made man pages for logformat, and all new commands
                  Added +-*/ to SPD(OPCS): OK
                  Added '-all' flag to SHOW(OPCS): OK

k1.12d - 04/07/98 BUGFIX: pro2display disables ratio too
                  BUGFIX: Enabled runbar() again (been off how long?!)
                  Status line is now:
                     <-0 RUNBAR 24-><-25 SCRIPT 52-><-53 FADE/DX 79->

k1.12c - 04/02/98 added defs: pro2display, set/clr/xorbit
k1.12b - xx/xx/9x added 'logformat' command
```

```
k1.12a - xx/xx/9x added feet/frames to logcounters
k1.12  - xx/xx/9x FINAL FIX for stupid screen scroll problem (mdrive.asm)
k1.11e - xx/xx/9x -x flag added to fdi/fdo/dxi/dxo
k1.10e - xx/xx/9x rat 1 0 1: fix to run in tandem
k1.10d - xx/xx/9x key: projector run keys don't check buckview now.
k1.10c - xx/xx/9x Bugfix for (176'-32) vs (176'0-32) in evalnum.
k1.10b - 08-06-92 key wont cls after called from script
k1.10a - 07-24-92 ALT-letter works a little better.
k1.10  - 05-29-92 Added softlatch to handle kuper's I/O port card
                  (which can be written to, but not read). Added
                  'rat 4 4' to shoot 4, advance 4, shoot 4, etc.
k1.09  - 04-22-92
         SPD BUGFIX: opcsdefs wasn't passing speed scale to kuper subs
         BIG BUG FIX --> in shutter: ContShut(): slow speeds caused
                  no ramping (ok), but an allstop failed (stopped out
                  of home) because rdsamp/allstop checking weird.
                  Put allstop opportunity AFTER vels get sent.
         ALSO: ShutterStop() wasn't checking for non-ramp case
               properly, and incorrectly computed if in midst of
               rampdown (added /32, and more reliable check. TESTED)
               Symptoms were stalls.
         ALSO: ContShutter() wasn't pointing ASADDR to velstop stack
               values (non-ramp), caused intermittent stall if ASTOP
               occurred during ShutterFlush() (which uses ASADDR if
               someone hits allstop)
k1.08  - 04-21-92 prophase added, spd(DEFS) handles '-' for any args.
k1.07  - 04-13-92 Added frange(OPCSDEFS) command at Tony's request.
k1.06  - 04-08-92 Bug fix in motorsubs.c: RampDownNow():
         ONTHEFLYRAMPDOWN. Bug fix: JOG: crawl FWD/REV now updates
         large display, improved bar() subroutine for faster display.
k1.05  - Added keydef(OPCSDEFS), JogRelease() etc.
k1.04  - is_all_stop_key() checks for allstop key even if 'motors off'.
         execute_args() now checks custom commands for all arguments.
k1.03  - Added 'respond', centralized the ALLSTOP handlers to the
         subroutines in OPCSSUBS, put in IFDEF OPCS clause for code
         in motorsubs.c, Evalnum: hex() function added,
         opcsjog: key->changed=0 when called. Seek added to KEY(OPCS).
k1.02f - run()'s ReadLine() modified: 199 changed to MAXLINECHARS,
         ReadLine() modified to handle 'Line too long' errors.
         Added ALT+chan letter to JOG.
         Fix run() to break when skipping if EOF reached.
         Jog comes up with 'D' as default instead of 'A'

k1.00 - This is a complete rewrite for use with kuper card and EMC subs.
        Customers wanted a microstepper version for their precision printers.

/|\
 |
K100 MICROSTEPPER

==                                    ====================================
== k1.00 - Kuper Microstepper Version ====================================
==                                    ====================================

HALF STEPPER
 |
\|/

3.00f - OPCSDEFS.C modified out redundant 'IsBadMotor()' calls.
3.00e - EFW Bugfixes:
        o PANIC/ABORT in a RUN file would continue anyway
          with next line. err=execute_string() in OPCSCMDS.
3.00d - REP >54 and REP PRO >54, added fpf_eval_num_expr(), fixed
        'K' in bignum.
3.00c - Fix:'motors off cam 12 motors off' screws up old counts.
        Added $1 $2 variable passing to RUNCMD.
        VCE's "102 PRINTER" RELEASE.
3.00b - Fix to seek() [seek 1 1 1 then seek 1 would run 1 1 1]
3.00a - JOG has inching, frame windoffs. (needs set-keys, slaving)
        Motor routines called w/func ptrs for counter updates. Slop
        routine bugfix. opcs has arg for different OPCSDEFS.OPC file.
3.00  - 12 channels supported, added +/- ratio to KEY. MANY subroutines
        have been modified in OPCSRAMP.C, and allstop() handling was
        greatly modified. Fix to SlopCorrection(). Fix to EvalFrameSpec:
        changed (int) typecast to (Frame). Added counter/rat sets to
        key_cmd, et al. Put back load_cmd.
2.10f - fixes to opcsinterp.c Lookup2Steps(using a [high] value would
        produce huge numbers) and to opcsramp (0xff instead of 0xffff).
        CS_WAIT has STI now. (bug came up in 103.00)
2.10e - timeinterp uses seconds/cycles, feed allows floats for interp
        channels.
2.10d - fixes to opcsinterp.c (free samp), jog added, fix ppr=1 slow
        added * for a chan spec, Chan2Bin() 'total' now a ptr, bug
        fix in GetFaderValue() to check for defs->interp==NULL,
        res2,rat2,seek2,chk2 have been phased out.
2.10c - floor(), fixes to step_cmd, opcsline.c, opcseval.c
```

```
2.10b - feed, step uses interps (ffocus, etc), PreCamEvent()
2.10a - timeinterp uses pps instead of seconds
2.10  - Large model, float interps, time interp, all 8 motors run
2.01b - buckle check BEFORE running motors, load removed
        VCE's "103 PRINTER" RELEASE
2.01a - key, load, lineup, less runbar calls, step abc no counter
2.01  - ramping, fast seek, pro2display bugfix, dirxor bugfix
2.00c - fix to SEEK..ShootRatio() contained wrong ratio
2.00b - fix to OPCSBIG (added NULL in pattern[] init), dutycycle, deen
2.00a - fix to shutdrvr.asm for projectors going 1/2 out of phase
2.00  - Parallel/steps/direction, 8 motor max, concurrent shooting.
1.50  - Commands and code added for DUAL projectors.
1.40a - Fixes to allstop handler, and DBXSAVE->COUNTER is now 'int'
1.40  - Lock command added, check buckle during shoot, allstop/buck
        checked in C, DBX saves/restores counters.
1.38  - Local/Public motor subs, contrived msg, (needs init code)
1.37d - step modified with optional arguments,log handles #-----
1.37c - better /etc/crash handling, LPT1/LPT2 hardware checks
1.37b - cam >-2 bugfix, debug info commented out (ONEIL 1/3/89)
1.37a - spacebar for allstop, double_check_hardware().

VERSION HISTORY
---------------
    APPLE ][+ OPCS: 1985 - 198x
    ---------------------------
    OPCS was initially conceived in the mid 1980's while I was a
    student at Calarts (82-86). Written in APPLE ][+ BASIC and 6502
    Assembly, this version was used by students until the late 80's.

    HALF STEPPER (PARALLEL PORT) OPCS
    ---------------------------------
    In 1988 Pat O'Neil asked if I could provide OPCS for one of his
    printers. The software was rewritten on the IBM PC in C and 8086
    assembly. This version built with the Microsoft V1.0 compiler,
    and was used by Pat O'Neil (Lookout Mountain Films) and Pete Kuran
    (VCE), and
    replaced the Apple ][+ system at Calarts.
    , using the parallel port to drive the motors dirctly.

    MICROSTEPPER (KUPER) OPCS: 1.xx
    -------------------------------
    When Bill Tondreau started Kuper Controls, he and I worked a deal
    where I could use his RTMC16 microstepper pulse generator cards
    on the PC to drive motors instead of parallel ports.  This was
    used on Pat O'Neil's second printer, with Centent microsteppers.
    This became the "K100" version of OPCS.  This version was licensed
    to many companies; Title House, Introvision, Technicolor, YCM,
    Cinetech, etc, etc. This version supports the RTMC16, RTMC48, and
    Kuper Industrial cards using different DOS TSR drivers.

KUPER OPCS: 2.xx
    During the world pandemic of 2020, I ported the code to Turbo C 3.0,
    and also developed my own microstepper card, the A800. This version
    continued support of all the above Kuper cards and the new A800.
    Turbo C had better error checking and modern C prototyping.
```



http://seriss.com/opcs/images/opcs-lineup-chart-6k-2024.png