THE UNIVERSITY OF MICHIGAN
COMPUTING RESEARCH LABORATORY

PDS USERS' MANUAL: INTRODUCTION

Edward Delp
CRL-TR-15-83

MARCH 1983

Room 1079, East Engineering Building
Ann Arbor, Michigan 48109
USA
Tel: (313) 763-8000

1Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors.
A software picture processing system (PPS) has been developed to aid research in image processing and pattern recognition. Most of the documentation is listed under PDS for Picture Data Structure which was the original name used for the picture data format. The system was designed at Purdue University to achieve the following goals: (a) provide users with an efficient set of programming tools for developing image processing algorithms, (b) provide easy access to special image peripherals and (c) make available a library of standard image processing functions.

PDS programs are called Image Processing Modules (IPM's). They may all be used as filters and have a special PDS syntax for specifying parameters. Much of the flexibility of IPM's arise from the way file names may be specified; it is strongly recommended that the file-names section of the documentation is read before attempting to use the system. A very useful program is display which enables image data to be written to an image display device with a minimum of effort.

Image Data Format

Image data in the PPS system may be formatted or unformatted. Unformatted data is simply a raw data file; the user must explicitly specify the format with the file name when processing this data with an IPM.

Formatted data has a header block of 512 bytes which precedes the raw data. The format of the data is specified by five integers in the header block:

x: x dimension (width)
y: y dimension (height)
h: number of bits per pixel
n: number of channels (for multispectral data)
f: format number (pixel format)

The header also contains two character strings of descriptive information called the title and the description. The title may be changed by an IPM but the description is always passed unchanged. Many IPM's use the command character string for the new title. In this way, it is often possible to see what operations have been applied to a processed image by printing the title. The description may contain any information which the user would like to have maintained with the data. This information will then be available with any processed image file which is derived from the original annotated image file. Other information maintained in the header includes a PDS file identifier, which permits
IPM's to check for formatted data files; the creator identifier and the creation date. Some formats for other systems, e.g. the NATO format, may contain a longer header with additional information. This information is stored after the data section of the PDS file when it is read from tape with an IPM, but it is not passed to any processed files. For details of the PDS header see /usr/include/pds.h.

Many data formats are available from single bitplane to multispectral floating point data, however IPM's usually only support one or two formats (8 bit unsigned character is the most common). To alleviate this problem, some format converting IPM's are available. New formats may be easily added to the system by allocating a new format number.

Implementation

The system was originally written in C (version 7 UNIX) for the PDP-11. It was converted to run on the VAX 11/780.

Programming Tools

IPM's may be written in either Fortran or C. The programming tools are available in both languages. PPS data files may be accessed either directly, which is the usual mode for Fortran and C programs, or with run-time parameter checking which aids error detection and program development.

For direct data access, procedures for opening the data files and parsing the argument character string are available. The output header is created and structures containing the header information for all files are made available. It is the responsibility of the programmer to perform all format compatibility checks and to manage the I/O and data processing.

In the error checking scheme all access to image data is made through PPS run-time library functions. Routines are available for reading and writing a pixel, a row, a column or a block of data. Each routine requires the x and y location, size and format of the data to be specified with each call and these parameters are checked at run-time. This error checking may be a significant part of the processing time if I/O is done at the pixel level but is usually very small for the other access modes. The "-lp" library must be linked. See the runtime-lib section for details.

Many image processing algorithms involve a computation over the local neighborhood of a pixel. A special PPS program has been written which efficiently scans an image and makes the local neighborhood available at each pixel position. The size of the window may be specified as an IPM parameter. To program a new window IPM only the algorithm which processes the local neighborhood and the format of the data needs to be specified, the rest is already programmed. New simple window functions can be programmed and installed in the system as an IPM in just a few minutes. The "-lpw" library must be linked. See the window lib section for details.
Acknowledgements

Prof. Anthony P. Reeves of Purdue University (now associated with Cornell University) is the "father" of the concept of PPS/PDS. Many other people have been involved in writing IPM's. Some of the major contributions at Purdue have been made by Jim Besemer, John Bruner, Mike Zuhl, Mark Fisher, Luciano Dalleore and Doug Kimber. George Gobel and Bill Croft developed the network system at Purdue which enabled distributed processing to be achieved. At the University of Michigan the major contributors were Arlan Martin, Gil McRAFT, Doug Kimber, and Jim Poje. Other programmers have also made contributions, the documentation indicates the author's name where possible.

If you have any IPM programs which may be of use to others, please communicate them (with documentation) to Prof. Edward J. Delp (login "ed") and they will be distributed.
PPS (PDS) USER'S MANUAL

CONTENTS

Introduction

File-Names

file-names - Miscellaneous information about PDS files

Image Processing Programs

bptobyte - convert from bit-plane to byte format
bytetobp - convert from byte to bit-plane format
combine - Combine many files into 1 multispectral file
compare - Compare two images
display - Puts a picture on the Ramtek or DeAnza
embed - Embed an PDS file onto another PDS file
enhance - deblur a picture
enlarge - Expand a PDS picture by an integral factor
extract - Take a subset of a multi-channel PDS file
fix - convert the input data to byte format
fmap - Map a picture according to a function map
hfft - fft on the rows of an image
hist - histogram plotting program
insert - Insert a PDS file into another PDS file
pdshc - put a picture on Printronix line-printer
reduce - Reduce image by an integral factor, uses mean
resample - Change image size by bilinear interpolation
reshape - Change the size of a picture by ANY factor
shrink - Shrink a PDS picture by an integral factor
stretch - stretch an image to fill range from 0 to 255
trans - Transpose a PDS picture on any of the 3 axes
trim - extract a portion of a pds image
vp - Put a picture on the Benson-Varian plotter
window - (local window programs)
ccs - Robert's cross gradient program
lthin - performs a line thinning algorithm
mean - computes the local mean
median - computes the local median
meddev - computes median deviation from the median
sdev  - computes the local standard deviation
sobel - sobel edge detection program
cmedian - leaves pixel or replaces with median if outside range
qmedian - performs median filtering within a percentage
cf1   - full Frei-Chen edge detection
cf2   - Frei-Chen isotropic gradient edge detection
cf3   - Frei-Chen Laplacian edge detection

Arithmetic programs
add   - add two image files
and   - bitwise and two images, one may be a bit-plane
div   - divide one image by another
invert - bitwise invert an image
mul   - multiply two image files together
or    - bitwise or two images

PDS header manipulation programs:
pdedit - Edit a PDS picture header
list   - Prints PDS file header on the terminal
mkpds  - Creates header for making PDS files
raw    - Remove the header from a PDS file
title  - Replace the title in a PDS file header

PDS runtime library routines:
runtime-lib  - Information about PDS runtime library
window-lib   - Information about window runtime library

LARS Multi-channel Analysis programs:
(not available at this time)
classify      - Run a maximum likelihood classification
cluster       - Perform an iterative clustering
hist          - Generate histograms for multi-spectral data files
laread        - Generate a PDS file from a LARS "MIST" type file
stat          - Compile various statistics about a "classify" run

Nato format:
(not available at this time)
natocat       - catalog a nato format tape
natoread      - generate a pds file from a nato format tape
natawrite - generate a nato format tape from a pds file
NAME

file-names - PDS file name syntax

SYNOPSIS

[ file ][ .raw ][ .read ][ .n ][ .x=n ][ .y=n ][ .z=n ][ .c=n ][ .f=n ][ .t="title" ]

DESCRIPTION

The PDS utility programs (e.g., shrink, trans) take as input and produce as output picture files that have a 512 byte header known (strangely enough) as the PDS header. This header contains information about the data, including picture size, format, bits per pixel, number of channels, and title. (This information may be listed by "list".) To add flexibility, modifiers are allowed after the file name separated by commas. These modifiers replace or alter the PDS header. In particular, their use allows these PDS routines to read and produce non-PDS format files.

If the file name is null, then standard input or output is assumed, depending on whether the file is being used for input or output. If there are no modifiers the file is assumed to be in PDS format and (for the read case) the header is read in, and in the write case a header is produced.

The RAMTEK and DEANZA must be treated specially by the PDS programs because no header block should be read from or written to it. Therefore, references to these two devices are recognized and handled correctly. In addition, file names of "0", "1", "2", and "3" are abbreviations for "/dev/dea0", "/dev/dea1", "/dev/dea2", and "/dev/image", respectively.

If present, modifiers replace or alter the header. In any case, a header with all the information is generated and used internally to the program. Modifiers are of the form:

  <modifier>

i.e., separated from themselves and the file name by a comma. This means that a file name may not contain a comma. The modifiers may be any of the following, and in the case of conflicts the most recently used take precedence.

read (read access files only) causes the header to be read and following options to override it. Must occur before any other options.

raw (write access files only) causes new file NOT to have a header.

nnn where "nnn" is a number. This sets both the X and Y sizes to be "nnn".

x=n where 'n' is a number. Sets the X size to "n".

y=n Sets the Y size to "n".

f=n set format type to "n" (an integer).

z=n set Z size (bits per pixel).
c=n      sets number of channels.

t="title" sets title as indicated. Defaults to command line. Note that title
may not contain a comma and any funny characters must be escaped. The title is ended by a comma or a zero char. In the write case where
there is no title specified, the command line is reproduced the same
way that "ps" does and used for a title.

DEFAULTS
When the header is not read and not all of the possible modifiers have been
mentioned, those parameters that require values are assigned the default
values. These are the defaults, although they may be overridden by some
routines.

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>DEFAULT VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>X size (x=)</td>
<td>512</td>
</tr>
<tr>
<td>Y size (y=)</td>
<td>512</td>
</tr>
<tr>
<td>format (f=)</td>
<td>1 (&quot;data&quot;)</td>
</tr>
<tr>
<td>bits per pixel (z=)</td>
<td>8</td>
</tr>
<tr>
<td>number of channels (c=)</td>
<td>1</td>
</tr>
</tbody>
</table>

EXAMPLE
Suppose that we want to use file "/pix/usuc/girl" as input to a PDS program,
but that that file is not in PDS format. Since the file is 256x256,
/pix/usuc/girl,256
would describe it entirely.

Note that by virtue of the defaults, a non-PDS picture of the size 512x512 may
be referred to by simply appending a comma (,) to its name.

BUGS
All PDS programs will not accept temporary files beginning with the # symbol.
NAME
bptobyte - change from bit-plane to byte format

AUTHOR
Doug Kimber

DATE WRITTEN
6/80

SOURCE LANGUAGE
C

SYNOPSIS
bptobyte if=infile of=outfile [-ms]

DESCRIPTION
This program takes the input file infile in bit-plane format and converts it to byte format in the output file. If there are less than eight planes in the input file (specified by the zsize of the input file) then the most significant bits of each byte will be zero filled to create 8 bit pixels. The -ms option causes the least significant bits to be zero filled instead of the most significant to create eight bit pixels.

BUGS
NAME
bytetobp - convert from byte data to bit-plane format

AUTHOR
Doug Kimber

DATE WRITTEN
8/80

SOURCE LANGUAGE
C

SYNOPSIS
bytetobp if=infile of=outfile [-cp]

DESCRIPTION
This program takes the input image file and converts it from byte to bit-plane format. If the -cp option is specified the program will calculate how many of the most significant planes are entirely zero, and output only the other planes, beginning with the most significant non-zero plane.

BUGS
NAME
   combine - many files into 1 multispectral file

AUTHOR
   Doug Kimber

DATE WRITTEN
   7/80

SOURCE LANGUAGE
   C

SYNOPSIS
   combine i1=file1 i2=file2 i3=file3 ... of=outfile

DESCRIPTION
   The current version of combine will accept up to ten input files and combine
   them into one multispectral file. Each file will be mapped to the channel number
   that is the same as the file number, eg. - i3 will be mapped to channel 3.

BUGS
NAME
compare - compare two images

Doug Kimber

DATE WRITTEN
7/80

SOURCE LANGUAGE
C

SYNOPSIS
compare if=file1 ig=file2 of=outfile [-dif] [-sdif] [-ivdif]

DESCRIPTION
This program will compare two input files on a pixel by pixel basis. If the -dif option is specified then the absolute value of the difference of the two images will be generated. The -sdif option will cause the absolute difference of the squares, divided by 255, to be generated. The -ivdif option will cause the absolute value of 255 minus the difference of the two input images to be generated.

BUGS
NAME
display - display a picture

AUTHOR
Mike Zuhl

DATE WRITTEN
8/77

SOURCE LANGUAGE
C

SYNOPSIS
display [-vfr] file [commands ... ]

DESCRIPTION
Display, among other things, puts a picture on the RAMTEK or DEANZA. The program itself doesn’t dirty its hands by playing with actual pictures, rather it invokes various PDS utilities to do its bidding.

The "v" flag causes all of its actions to be displayed. The commands are displayed in almost identical manner as they would be input to the shell, except for the normalizing (see below). File deletions are shown as "rm filename".

The "f" flag causes display to use temporary files instead of pipes. In addition, specifying the "f" flag causes all the programs to run serially, rather than in parallel, and should take longer. The "f" flag should seldom be used because the generated files are usually very large and run a serious risk of overflowing some device, for which there is no check. The only reason that the "f" flag was implemented was because we were having troubles with pipes for a while. Now, about the only reason to specify the "f" flag is when you are reading and writing on the same image. Note: if you interrupt display (e.g., via the "delete" key) it will clean up all the scratch files.

The "r" flag causes display to instruct all size changes to be done in floating point ("real") mode. This makes the pictures more pleasing to look at, but you can’t see the individual expanded pixels (a slight loss, sometimes) and it takes longer. Compare it both ways to see which you prefer.

An input file is required. If the file name is "-", standard input is used. If the input file is not in standard PDS format, the user is prompted to supply its size. There is no such recovery for pipes. If you want to pipe a non-PDS picture, use a null file name and PDS modifiers.

Special care must be taken when reading from devices, because although display does not process pictures, it does attempt to read the PDS header if there should be one. In particular, if you try to read a PDS format file from tape with rewind disabled, at the very least you will lose the first line and it probably won’t work at all (and not tell you why). (See "help pds/file-names").

The following keyword commands are processed in order. If the last
parameter is not an output file name, the output is written onto the RAMTEK image.

part <position> <size>
   Extracts the part of the picture whose upper corner is <position> and whose size is <size>. <position> is in the form of npn, where the "n"s are integers indicating, respectively, X and Y coordinates in pixels. The <size> parameter of a similar format, except that instead of a "p" for a separator, an "x" is used. <position> and <size> may be interchanged or defaulted. <position> defaults to centering the picture and <size> defaults to 256x256.

section <position> <size>
   Identical to "part", above.

trans
   Or "transpose". Transpose picture along the diagonal. For large pictures, this can be slow.

vtrans
   Or "transv" or "vt". Transpose along the vertical axis.

htrans
   Or "transh" or "ht". Transpose along the horizontal axis.

into <position> <file>
   Or "insert". Inserts a picture into an already existing picture, with its upper left corner at <position>. See "part", above. If <file> is missing, it defaults to the RAMTEK. <position> defaults to centering the picture.

center <file>
   Or "center". Center the picture (strangely enough). <file> defaults to the RAMTEK

size <size>
   Change the size of the picture to <size>. See "part", above, for a description of <size>. The aspect ratio of the picture is preserved by filling the top and bottom, or left and right, by zero background. If no <size> is specified, this command is ignored.

shape <size>
   Or "reshape". Similar to "size", above, except that it does allow the aspect ratio to change.

<file>
   A parameter not recognizable as one of the above keywords is considered to be a file name and the current picture is written to it. Note that this is a PDS file name and as such can have any of the PDS modifiers. If there are commands following they will then use this file as input. This allows intermediate pictures to be saved.
Commands that take a <file> (i.e., "into" and "center") do not check to see if the parameter is a keyword. If it's there, they take it.

EXAMPLES
The command

    display myfile 0

will cause "myfile" to have its size adjusted to fill the screen and placed on the DEANZA image 0.

In the following examples, the images are displayed on the Ramtek.

Suppose we want to take the file "myfile", shrink it down to 50x50, and put it into Ramtek at 450p35.

    display myfile size 50x50 into 450p35

Now we want to do the same thing, only elongate the inserted picture twice in the X direction.

    display myfile shape 100x50 into 450p50

Notice the use of "shape" rather than "size". If "size" had been used, there would have been a black box to either side of the embedded image.

SEE ALSO
    reshape
    trim
    trans
    insert
    embed
    file-names

FILES
    /tmp/display.[a-z][0-9]*          scratch files with "-f"

BUGS
    If something goes wrong in the middle of a command, display doesn't know and thus can propagate a messed up file.
NAME
embed - embed one pds format file within another

DATE WRITTEN
7/77

SOURCE LANGUAGE
C

SYNOPSIS
embed [-c] +x,y if=infile ef=embedfile of=outfile

DESCRIPTION
Embed takes an embedfile and "overlays" it over the background of infile. The upper lefthand corner will be at "+x,y". The default positioning is to center the picture.

The "-c" (center) flag causes all zeros to be used instead of reading the "if=" input file. This feature replaces the "center" program which would center a picture on the RAMTEK or DEANZA.

SEE ALSO
pds/file-names

BUGS
NAME
  enhance - deblur a picture

AUTHOR
  Tim Rinker

DATE WRITTEN
  6/81

SOURCE LANGUAGE
  C

SYNOPSIS
  enhance [-c] if=infile of=outfile [wd=[5, 9]]

DESCRIPTION
  Enhance uses the laplacian operator to deblur an image. The -c option will print
  out the number of enhanced pixels with value greater than 255 that were set to
  255 and the number of enhanced pixels with value less than 0 that were set to 0.
  The user may specify a window with a weight of 5 in the center, -1 at the 4
  nearest neighbors and a 0 at the other neighbors. This is the default window.
  The other window has a weight of 9 in the center and and -1 at the 8 neighbors.

EXAMPLES
  enhance -c if=pix1 of=pix2
  enhance if=pix1 of=pix2 wd=9

BUGS
NAME
enlarge - expand a PDS format picture by a given factor

AUTHOR
Mike Zuhl

DATE WRITTEN
7/77

SOURCE LANGUAGE
C

SYNOPSIS
expand [+x,y] if=infile of=outfile

DESCRIPTION
This program expands a PDS-formatted picture by an integral factor, denoted by "+x,y". The default expansion size is 2 in both the X and Y dimensions.

BUGS
NAME
extract - extract channel(s) from a multi-channel PDS file

AUTHOR
Mike Zuhl

DATE WRITTEN
8/77

SOURCE LANGUAGE
C

SYNOPSIS
extract ch=n,n,... if=infile of=outfile

DESCRIPTION
The program takes a multi-channel PDS file as input and writes a subset of the channels as a PDS output file. Infile defaults to standard input and outfile defaults to standard output.

The "ch=" parameter is a list of channels from one to the number of channels on the input. The channels may be in any order and may be repeated. For instance

    extract ch=1,1,1 if=in of=out

will replicate channel 1 three times. The channel selection defaults to "ch=1".

EXAMPLE
Suppose we have a three channel file, "zap", and we want to reverse the order of the channels.

    extract ch=3,2,1 if=zap of=zip

will do the trick.
NAME
  fix - change format of input file to byte

AUTHOR
  Doug Kimber

DATE WRITTEN
  7/80

SOURCE LANGUAGE
  C

SYNOPSIS
  fix if=infile of=outfile [-s]

DESCRIPTION
  This program takes infile in integer, 32 bit floating point, or 64 bit floating point
  and converts it to byte format. The -s option will cause the output not to be
  scaled.

BUGS
NAME
fmap - map image using function memory map of the Ramtek

AUTHOR
Jim Besemer and Mike Zuhl

DATE WRITTEN
7/???

SOURCE LANGUAGE
C

SYNOPSIS
fmap [if=...] [of=...] [fm=...]

options:
if=xxx use file xxx for input instead of /dev/image
of=yyy use yyy for output file (default = standard output)
fm=zxx use zzz for function memory file (instead of RAMTEK lookup table).

DESCRIPTION
Fmap copies a file, translating the gray-levels according to some Ramtek function memory mapping. The defaults are set up so that the input (both image and mapping) is from the Ramtek, and the output is a file. It may be used, however, to apply an arbitrary mapping to any file.

EXAMPLES
% fmap of=xxx copy image from /dev/image and put it into file xxx
and map through RAMTEK lookup table

% fmap if=yyy of=zxx copy file yyy to zzz using the mapping present in RAMTEK lookup table

% fmap if=yyy of=zxx fm=fff copy yyy to zzz using mapping in file fff.

SEE ALSO
fmem
NAME
    hfft - fft on the rows of an image

AUTHOR
    A. P. Reeves

DATE WRITTEN
    10/78

SOURCE LANGUAGE
    C

SYNOPSIS
    hfft [-i] [-m] if=infile of=outfile

DESCRIPTION
    This program will perform the fft or the inverse fft on the rows of an image. The
    -i option specifies the inverse transform. The -m option inhibits the automatic
    modulation of -1**i*j. This routine can deal with pipes.

BUGS
NAME
hist - plot distribution of pixel values

AUTHOR
Dave Olander

DATE WRITTEN
July 1981

SOURCE LANGUAGE
C

SYNOPSIS
hist if=infile [arguments]

DESCRIPTION
This program plots a graph of pixel values vs. the number of pixels with those values, for the input image. The graph may be displayed on various graphic devices. The following options are available to modify the graph:

-p The plot is written out to a file 'graph'. This file can then be written out to the Versatec using GP or to the Printronix line printer using GPLP. This is the default source of the plot.

-o Same as -p except the graph is written out to standard output. This is helpful if the '-i' option is used with GP and GPLP.

of=outfile The file that the plot is saved in using the -p option is changed to 'outfile'.

-n Display the plot on the Ramtek's graphic overlay n, where n can be 0 or 1. The default value is 0.

-g Display the plot on a Ramtek graphic overlay.

-r Display the plot on a Ramtek image.

-t Display the plot on a Tektronix 4010 or 4014 display.

-h Display the plot on the HP plotter.

-b The display device is not blanked before plotting. This has no effect on the HP plotter.

-z A bar graph is plotted instead of connecting points with a
straight line.

-\(c\)  
Plot the cumulative distribution of the pixel values instead of the normal plot.

-\(e\)  
Calculate the entropy of the plot. The entropy value is displayed with the plot and is printed to standard output.

-\(l\)  
List the pixel distribution (i.e. pixel value and number of pixels with that value) on standard output.

-\(s\)  
Scale the plot between 0 and 1.

SEE ALSO  
gp, gplp

BUGS  
At present, it is not possible to display plots on the HP plotter.
NAME
insert - insert a picture into an existing picture

AUTHOR
Mike Zuhl

DATE WRITTEN
8/77

SYNOPSIS
insert [-XY] if=infile of=outfile

DESCRIPTION
This program inserts a file in another. It is similar to embed, except that it
puts infile into outfile rather than creating a third file. It is used mostly for
the RAMTEK and DEANZA.

XY is the position of the upper left corner in outfile to insert infile. It defaults
to centering the picture.

BUGS
NAME

pdshc - print pds file on printronix

AUTHOR

Mark Diamond and Gilbert McGrath (University of Michigan)

DATE WRITTEN

8/81

SYNOPSIS

pdshc if=inflie [l=table] [-m]

DESCRIPTION

This program takes any input image and reshapes it to a 256x25 image maintain-
ing the aspect ratio and prints the image on the printronix.

The program by default lightens the image by a non-linear transformation mapping the pixel values 0-255 into a range of 0-15.

The [-m] option maps the pixel values 0-255 into the range 0-15 linearly.

An optional look up table may be inputed [l=table] if the user desires to create a different mapping. The table must be 256 short integers in the range 0-15, 0 being the darkest and 15 the lightest.

SEE ALSO

reshape
NAME
reduce - reduce an image by an integral factor

AUTHOR
D. A. Kimber

DATE WRITTEN
7/80

SOURCE LANGUAGE
C

SYNOPSIS
reduce if=infile of=outfile [xs=nnn] [ys=nnn]

DESCRIPTION
This program takes the input image and reduces it by an integral factor in both the x and y dimensions. The output image is created by taking the mean of a window of size nnn by nnn. nnn is the factor by which the image is to be reduced in the corresponding dimension. The default reduction is 2 in each direction. It is an error if the factor does not divide the dimension evenly.

BUGS

SEE ALSO
shrink
NAME
resample — reduce (enlarge) an image to a specified size

AUTHOR
D. A. Kimber

DATE WRITTEN
7/80

SYNOPSIS
resample if=infile of=outfile [xs=nnn] [ys=nnn]

DESCRIPTION
This program will change the size of the input image to be the size specified by
the parameters xs=nnn and ys=nnn. Default is square if only one of xs or ys is
specified. At least one must be specified. The output is generated from the
input using a bi-linear interpolation algorithm.

BUGS
NAME
reshape - Change picture to arbitrary size.

AUTHOR
Mike Zuhl

DATE WRITTEN
August 1977

SOURCE LANGUAGE
C

SYNOPSIS
reshape [-a][-f] if=infile of=outfile

DESCRIPTION
Reshape changes the size of a picture to any other size. It differs from the programs shrink and expand in that reshape allows you to change the size by any factor while shrink and expand are limited to integral factors. Also, reshape will allow multi-channel files and some other funny types (like fortran complex) to be manipulated.

If the "-f" option is supplied, reshape uses floating point interpolation rather than dropping or repeating pixel values. This means that 1) pictures look more realistic, and 2) it takes about twice as long. The default, integer mode is much faster and therefore usually preferable.

The output size is determined by the PDS modifiers put on the output file. Note that when writing to the RAMTEK or DEANZA these modifiers are magically supplied for you. If the input size is the same as the output size, a straight copy is done.

Ordinarily, reshape preserves the aspect ratio by providing a black (0) background either at the top and bottom, or the left and right. That is,

% reshape if=a,x=100,y=50 of=b,x=200,y=200

would cause file b to contain a 200x200 picture with a 50 pixel wide black stripe on the left and the right. This is to keep from inadvertently distorting the picture. The "-a" option overrides this feature and causes the output picture to fill the image, hang the aspect ratio. Very funny pictures can be made this way.

Infile defaults to standard input and outfile defaults to standard output, as you would expect. If infile is specified, you can drop if "if=" and/or the "of=" prefix.

SEE ALSO
enlarge
shrink
resample
reduce

**BUGS**

Slow, in floating point mode.
Very small images (e.g., 2x2) come out real funny.
NAME  
shrink - shrink PDS format pictures

AUTHOR  
Mike Zuhl

DATE WRITTEN  
7/77

SOURCE LANGUAGE  
C

SYNOPSIS  
shrink [+x,y] if=infile of=outfile [-max][-min][-mean][-rand]

DESCRIPTION  
Shrink is used to make a picture smaller by an integral factor. "x" and "y" are the X and Y shrink factors, respectively. If "x" is specified and "y" is omitted then both X and Y shrink factors are set to that value. The default is "+2".

"Infile" and "outfile" are the PDS format input and output files, respectively. They default to standard input and standard output. The four options max, min, mean, and rand determine how the output file will be generated. Max takes the maximum pixel in the window, min the minimum pixel, mean an average value, and rand a random pixel from the window. If none of these four options are specified the upper left pixel of each window will be chosen.

SEE ALSO  
reduce

BUGS
NAME
stretch -- pixel values to cover range from 0 to 255

AUTHOR
Doug Kimber

DATE WRITTEN
10/80

SOURCE LANGUAGE
C

SYNOPSIS
stretch if=infile of=outfile [ch=channel]

DESCRIPTION
This program normalizes the pixels in the specified channel of the input image to fill the range from 0 to 255 by subtracting the minimum, multiplying by 255, and then dividing by the difference of the max and min. "infil" and "oufile" are the input and output files, respectively, with optional modifiers. If the channel is not specified then channel 1 is assumed (a message to this effect is printed to alleviate the possibility of accidentally omitting the channel when 1 is not the desired channel).

BUGS
NAME
trans - transpose PDS picture files

AUTHOR
Mike Zuhl

DATE WRITTEN
8/77

SOURCE LANGUAGE
C

SYNOPSIS
trans [-v] [-h] [if=infile] [of=outfile]

DESCRIPTION
This program transposes PDS-formatted picture files. By default the trans-
spose is done on the diagonal axis, but it can also be done on the vertical ("-v"
option) or the horizontal ("-h" option) axis.

"Infile" and "outfile" default to standard input and standard output, respec-
tively. Since some of the operations may require seeks on a file temp files will
be made if necessary.

FILES
/tmp/trans.* temp files when used with pipes

BUGS
NAME
trim - extract a portion of a PDS picture

AUTHOR
Mike Zuhl

DATE WRITTEN
7/77

SOURCE LANGUAGE
C

SYNOPSIS
trim [-x,y] [xy=x,y] [+x,y] if=infile of=outfile

DESCRIPTION
Trim extracts a portion of a picture from another PDS format picture. The upper lefthand corner of the picture extract is denoted by "+x,y", or by "xy=x,y". The resultant picture size is normally specified as a modifier to the output file, but may also be specified as "-x,y". This defaults to trimming out the center. The size of the output picture is determined by the modifiers to the output file.

BUGS
NAME
vp — print a PDS picture on the Benson-Varian.

SYNOPSIS
vp [-n ] [ 1=levelfile ] if=infile

DESCRIPTION
Vp read a PDS format picture file and generates a 17 graylevel picture to be output on the spooler's output device: the benson varian. A pixel is represented by a 4 by 4 square of dots. The graylevels are produced by turning on from zero to 16 dots, for a total of 17 levels. Since there are a total of 2112 dots per line on the device, the maximum picture width is 528. Wider pictures are silently truncated.

By default, vp chooses a uniform mapping of the possible 256 input graylevels to the representable 17 output graylevels. However, this mapping is seldom ideal, so the "l=" parameter allows the user to specify his own mapping. "Levelfile", selected by the "l=" parameter, si a file containing 256 binary integers (words) in the range zero to 16 (integers outside this range will be noisily mapped into the range). The ith entry in the table contains the level to which the input level i is to be mapped. This is similar to the way in which the RAMTEK function memory works, and is the same format as produced by copying from /dev/fmem.

If "infile" is not specified, then it defaults to standard input. Since the input is a PDS file, modifiers can be used.

The "-n" option causes the negative of the picture to be produced.

FILES
vp is a csh script which feeds the output of the "nvp" program into "sp -rv".
/usr/spool/spool/loops*
/usr/ece/nvp
/usr/src/ece/cmd/spool/xxx
/usr/adm/spool_log

BUGS
17 levels aren't alot.

SEE ALSO
spdrop(1), sphold(1), spst(1), spq(1), sprm(1), bp(1)

EXAMPLES
Try "vp if=/pix/usc/monkey,512". Be patient.
NAME
local window IPM's

SYNOPSIS
<program name> [if=inputfile] [of=outputfile] [xs=xnn] [ys=ynn][d] [-str]
[help]

DESCRIPTION OF PARAMETERS
inputfile: the file the input is to be taken from
outputfile: the file the output is to be put on
xnn: the size of the window in the x direction Defaults to dsizex in window program.
ynn: the size of the window in the y direction Defaults to dsizey in window program.
d: set debug flag to help debug program. Used by support functions and may also be used in window programs to aid debugging.
str: any user defined string. The use of the specified options is program dependent.
help: will invoke the printing of help information from the window program. This usually describes the syntax for the input command and contains a brief description of what the program does. Use of the help option inhibits actual execution of the window program and only prints the help string.

GENERAL DESCRIPTION
These local window IPMs are a set of routines for efficient local window scanning. Local window IPMs involve a common set of support subroutines, and are, in general, very simple to write and use the above syntax. The following IPMs are currently available:

1) ccs - computes the Robert's cross gradient
2) lthin - computes a line thinning algorithm
3) mean - computes the local mean
4) median - computes the local median
5) meddev - computes median deviation from median
6) sdev - computes the local standard deviation
7) sobel - computes the Sobel function, isotropic or euclidean
8) cmedian - leaves pixel or replaces with median if outside range
9) qmedian - performs median filtering within a percentage
10) cf1  - full Frei-Chen edge detection
11) cf2  - Frei-Chen isotropic gradient edge detection
12) cf3  - Frei-Chen Laplacian edge detection

These existing programs are for one data type - byte data, but may be easily modified by a simple source program edit. See "how to write a window program" for details.

DESCRIPTION OF AVAILABLE IPMS

ccs

ccs performs the Robert's cross gradient function on the input file by using the formula p[i,j] = |p[i,j] - p[i+1,j+1]| + |p[i+1,j] - p[i,j+1]|. The default window size is 2x2, since other window sizes are nonsensical. The current program does not check to see if the size has been changed to other than 2x2. If the window goes outside the edges of the input data then all values outside the border are assumed zero. There is a possibility of overflow for a given pixel as the result for very sharp edges could require 9 bits to represent. No overflow check is made, and the overflow bit will be ignored.

lthin

lthin performs a line thinning algorithm by considering the 3 basic conditions for eliminating a pixel in all the possible rotations, in terms of the requirements for 0's. This is done by testing if there is a 0 in a position of the window. If a logical "1" is found, then all the combinations that cannot be realized are masked out. The basic conditions are shown here, with d=don't care, considered "1". The x is the pixel that will be removed.

\[
\begin{array}{ccc}
d0d & d00 & 00d \\
1x11x0 & 0x1 \\
111111 & 111 \\
\end{array}
\]

By using the "nn" option the logic "1" pixel value may be selected. The logic "1" default value is 64.

The default window size is 3x3, since this is the only window size that makes any sense. The current program does not check to see if the size has been changed from 3x3. If the window goes beyond the edges of the input data then all values outside the data border are assumed zero.

mean

mean computes the local mean of the given window. The default window size is 2x2. If only one of xs or ys is specified the window defaults to a square of the size
specified. If the window goes beyond the edges of the input data then all values outside the data border are assumed zero.

**median**

median computes the local median of the input window. Median defaults to a 3x3 window if xs and ys are unspecified. If only one is specified then the window defaults to a square of the size specified. If the window should overlap past the edges of the input data then all values beyond the data border are assumed to be zero.

**meddev**

meddev computes the median deviation from the median of the pixels in the window and sets the center pixel to that value. The window size defaults to a 3x3. If only one of xs or ys is specified then the window defaults to a square of that size. If the window overlaps the edges of the input data then all values beyond the data border are assumed zero.

**sdev**

sdev computes the local standard deviation of the input window. The window size defaults to 3x3. If only one of xs or ys is specified the window defaults to a square of the size specified. If the window overlaps beyond the edges ot the input data then all values beyond the data border are assumed to be zero.

**sobel**

sobel computes the sobel edge detection function. Use of the "-i" option will select the isotropic computation, while use of the "-e" option will select the euclidean norm computation. The default window size is 3x3. If only one of xs or ys is specified the window defaults to a square of the size specified. If the window overlaps the edges of the input data all values beyond the data border are assumed to be zero. The possibility of overflow does exist, as the result might require 9 bits to fully represent it. Currently, no overflow check is made, and the overflow bit will be ignored.

**cmedian**

This program selectively replaces the center pixel in the window with the median of the window, or its original value. The -nn option specifies that if the pixel falls within nn/2% of the median for that window then the pixel is left alone. Otherwise it is replaced by the median for that window. The default value for nn is 30%. (15% either side of the median).
qmedian

qmedian computes the local median of the input window. It defaults to a $3 \times 3$ window if x's and y's are unspecified. The center of the window is replaced by the median if it does not fall within a certain percentage. Default percentage is 30.

cf1

cf1 computes the Frei-Chen edge detection using the eight window transforms. Window size is $3 \times 3$. A threshold value is specified. Default threshold is 2.

cf2

cf2 computes the Frei-Chen edge detection using only the isotropic gradient transforms. Window size is $3 \times 3$. A threshold value is specified. Default threshold is 2.

cf3

cf3 computes the Frei-Chen edge detection using only the Laplacian transform. Window size is $3 \times 3$. A threshold value is specified. Default threshold is 2.
NAME
   add - add two image files

AUTHOR
   Doug Kimber

DATE WRITTEN
   6/80

SOURCE LANGUAGE
   C

SYNOPSIS
   add if=infile1 ig=infile2 of=outfile [-xn]

DESCRIPTION
   This program takes the two input images and adds them with optional modifiers.
   If the -xn option is selected then elements of infile2 are multiplied by n.

BUGS
NAME

and - bitwise and two images

AUTHOR

Doug Kimber

DATE WRITTEN

8/80

SOURCE LANGUAGE

C

SYNOPSIS

and if=infile1 ig=infile2 of=outfile [-bin]

DESCRIPTION

This program takes the two input images and does a logical and of their corresponding bytes. The -bin option specifies that all non-zero bytes of infile2 are to be taken as FF base 16. This is useful for anding a binary image with a regular byte image.

BUGS
NAME
div - divide one image by another

AUTHOR
A. P. Reeves

DATE WRITTEN
11/78

SOURCE LANGUAGE
C

SYNOPSIS
div if=infile1 ig=infile2 of=outfile

DESCRIPTION
This program divides the image of infile1 by the image in infile2 and puts the result in outfile.

BUGS
NAME
divc - divide one complex image by another

AUTHOR
A. P. Reeves

DATE WRITTEN
1/79

SOURCE LANGUAGE
C

SYNOPSIS
    divc if=infile1 ig=infile2 of=outfile [-s]

DESCRIPTION
    This program divides complex infile1 by complex infile2 and writes the result to
    outfile. The -s option is to give a result in the case of divide by 0. Otherwise a
    fatal error may occur.

BUGS
NAME
invert - take the ones complement of an image

AUTHOR
Doug Kimber

DATE WRITTEN
8/80

SOURCE LANGUAGE
C

SYNOPSIS
invert if=infile of=outfile

DESCRIPTION
This program takes the ones complement of each pixel of the input file and
writes it to the output file.

BUGS
NAME  
mul - multiply one image by another

AUTHOR  
Doug Kimber

DATE WRITTEN  
7/80

SOURCE LANGUAGE  
C

SYNOPSIS  
mul if=infile1 ig=infile2 of=outfile

DESCRIPTION  
This program will multiply the image in infile1 by the image in infile2 and place the result in outfile. Multiplication is performed on a pixel by pixel basis.

BUGS
NAME
   or - byte-wise or two images

AUTHOR
   Doug Kimber

DATE WRITTEN
   8/80

SOURCE LANGUAGE
   C

SYNOPSIS
   or if=infile1 ig=infile2 of=outfile

DESCRIPTION
   This program takes the two input images and does a logical or of their corresponding bytes.
NAME
list - list PDS header information about an image

AUTHORS
Jim Besemer and Doug Kimber

DATE WRITTEN
3/30/78

SOURCE LANGUAGE
C

SYNOPSIS
list pdsfile

DESCRIPTION
list will print out all the pertinent information that is contained in the header of
a normal PDS file. This includes the size of the image, title, description, creation
date, creator, PDS format, the number of channels, and any extended descrip-
tion that may follow the data.

DIAGNOSTICS
Usual messages that file is not in valid PDS format.

IMPLEMENTATION DESCRIPTION
list uses the format capabilities of printf to print out the information in the PDS
header. The description is blank filled on the left, so if there was no description
information, only 128 blanks will be printed.
NAME
mkpds - make a PDS file from a raw data file

AUTHORS
Jim Besemer, Doug Kimber, and Arlan Martin

DATE WRITTEN
3/30/78

SOURCE LANGUAGE
C

SYNOPSIS
mkpds [the program will prompt for all data]

DESCRIPTION
mkpds will create a new PDS image file by prompting for the necessary header
data. It requests the file name, number of data channels, the size of the image,
the PDS format (this can be an integer or string naming the type), the number
of bits per pixel, and a 40 character title. mkpds will then ask if there are any
other comments about the image.

FILES USED
/tmp/mkpdsxXXXXX temporary file
/usr/lib/pds/file.formats valid PDS file formats

BUGS
None known, except that it may leave files around if it is interrupted.
NAME
pdsedit -- edit a PDS file header

PROGRAM AUTHORS
Jim Besemer, Doug Kimber, and Arlan Martin

DATE WRITTEN
Unknown

SOURCE LANGUAGE
C

SYNOPSIS
pdsedit pdsfile [program prompts for parameters]

DESCRIPTION
pdsedit allows the information contained in the header of a PDS file to be altered. It can be useful during PDS program development, when the exact usage of header information is subject to change, and it is undesirable to regenerate the data files.

When the program is run, it prints an information message, and then waits for commands. If any error is detected by edit, it will ignore the rest of the line which caused the error. pdsedit is used interactively.

BUGS
NAME
  raw - remove the PDS header from a PDS file

AUTHOR
  Jim Besemer

DATE WRITTEN
  Unknown

SOURCE LANGUAGE
  shell file

SYNOPSIS
  raw if=infile of=outfile

DESCRIPTION
  Raw removes the PDS header from a file. It does this by merely copying the file
  but omitting the first block.

IMPLEMENTATION DESCRIPTION
  The entire shell file is as follows:

  dd $1 $2 skip=1

BUGS
NAME
title - put (new) title in PDS picture header

AUTHOR
Mike Zuhl

DATE WRITTEN
8/77

SOURCE LANGUAGE
C

SYNOPSIS
title "title" file(s)

DESCRIPTION
Title replaces the title in an already existing PDS file with the title specified. This is useful for re-commenting a file. Titles are also set by using the "t=..." modifier when creating a file. If this option is not specified, the default is the command line that created the file.

By placing modifiers on the files mentioned on the title command, you can also change some of the other information in the PDS header. Note that these changes affect only the header and do not touch the picture data itself.

BUGS
NAME

pds runtime routines

DESCRIPTION

Following is a summary of routines available in the PDS runtime library system. All the following routines (with the exception of those marked "internal use only") are available. Where necessary, the subroutine names have been truncated to 6 characters. The runtime library is linked using "-lp". Parsing of arguments is accomplished through the use of mparse as described in (A) below. Routines for direct access to files using UNIX file descriptors are described in section (B). Routines with run-time error checking, described in (C), use user specified unit numbers 0-15 to identify files (a concept similar to fortran unit numbers).

CONTENTS

(A) Parameter parsing

mparse - parses the parameter string

(B) Direct file access

opnpds - opens a pds file
popnpds - opens a pds file to work with pipes
clspds - closes a pds file, necessary when using popnpds

(C) Error checking access routines

getblk - read an arbitrary piece of a picture
putblk - write an arbitrary piece of a picture
blkio - (internal routine)
blkxfer - (internal routine)
readall - (internal routine)
pdsclose - clean-up and close a pds file
pdsexit - close all pds files and exit
getc - get a column from a pix
putc - put a column into a pix
getfmt - read format information from file header
putfmt - change or initialize file header
gettl - get pds title from header
putttl - change or initialize file header
itoa - convert integer to ascii
getline - get a line from pix
putline - put a line into pix
pdsipos - (internal routine)
pdsuok - (internal routine)
pdshok - (internal routine)
pdsbok - (internal routine)
pdsopen - open pds descriptor file
getpixel - get a pixel
putpixel - put a pixel
pdsppos - (internal use)
vfyfmt - verify pds format name
NAME

mparse

SYNTAX

mparse(argc, argv, par)

DESCRIPTION

This routine is passed the argument count (-argc-), the parameter list (-argv-), and an array of structures (-par-) and returns with the structure elements filled as specified.

The argument count and argument list are the same as is passed to the main routine by the system, and the format of the structure is {char *match; char *val;}

Parsing is done by examining each parameter in order and trying to match its initial substring to the supplied pattern for each structure not already matched. When a match occurs a pointer to the next character in the input line after the initial matched substring is set in the structure.

The array is ended by a zero match pointer. The number of matched arguments is returned as the value of the function.

The pointers to the matched string are cleared before parsing. Multiple occurrences of the same match string are allowed and will be filled in order of occurrence. A pointer to a null character (i.e. " ") will match anything.

This is useful for pulling off up to a fixed number of file names, for instance.

As an example, for the following array:

```
struct { char *m, *val;
    {par[]};
        "if=", 0,
        "of=", 0,
        "xf=", 0,
        " ", 0,
    0, 0;
```

and a command line of:

```
a.out if=infile of=outfile xf=23 -q
```

mparse would return with a pointer to the string "infile" in par[0].val, a pointer to the string "outfile" in par[1].val, a pointer to the character string "23" in par[2].val, and a pointer to the string "-q" in par[3].val. Normally par[3].val (for the given array) would be checked, and if non-zero an error message indicating an unrecognized parameter printed. Note that this must be done by the calling program.
NAME
cislps

SYNTAX
cislps(fd,name,access)

DESCRIPTION
This routine is necessary whenever popnpds is used to open a file with WANDSK access (4). The reason for this is that the contents of any temporary output file that may be created by popnpds must be copied into the pipe after execution. fd is the file descriptor of the file that was returned by popnpds, and name is a pointer to the name which the file was opened with. (Usually name will be an element of the array par used by mparse to parse the input line). The name itself is used only to check for null file names; if the file pointer is 0, or the contents of name are " " or "", then the output is assumed to be to a pipe and any temporary files copied to standard output.
NAME
opnpds

SYNTAX
fd = opnpds(name, header, access)

DESCRIPTION
This routine is internal to the system PDS utility programs and handles the
opening of files that may or may not be in PDS format. The call is:

    fd = opnpds( name, header, access)

Where:
fd                   the file descriptor for this file if >= 0.
                   the error code if < 0.
name                a pointer to the string containing the (optional) file name
                   and (optional) modifiers
header              pointer to the PDS header structure
access              read = 0; write = 1; read & write = 2;

A file name of the form "#2" (pound-sign followed by a number) specifies that
that file descriptor is to be used (assumed to already be open).
If the file name is null, then standard input or output is assumed, depending on
the access code. If there are no modifiers the file is assumed to be in PDS for-
mat and (for the read case) the header is read in. In the write case the element
"modified" in the header structure is set to one (zero otherwise). This means
that the header should be written on output only if this flag is non-zero.
If present, modifiers replace or alter the header. In any case, a header with all
the information is generated. Modifiers are of the form:
   <modifier>
 i.e., separated from themselves and the file name by a comma. This means that
a file name may not contain a comma. The modifiers may be any of the follow-
ing, and in the case of conflicts the most recently used take precedence.

    read       (read access files only) causes the header to be read and
               following options to override it. Must occur before any other
               options.

    raw        (write access files only) causes new file NOT to have a
               header.

    nnn        where "nnn" is a number. This sets both the X and Y sizes to
               be "nnn".

    x=n        where 'n' is a number. Sets the X size to "n".
y=n sets the Y size to "n".

f=n set format type to "n" (an integer).

z=n set Z size (bits per pixel).

c=n sets number of channels.

t="title" sets title as indicated. Defaults to command line. Note that title may not contain a comma and any funny characters must be escaped. The title is ended by a comma or a zero char.
NAME
popnpds

SYNTAX
popnpds(name, header, access)

DESCRIPTION
This routine is exactly the same as opnpds except that the access code may also be a 3 for read and seek, or a 4 for write and seek. If called with access 3 or 4 popnpds will create temporary files if the input (output) is coming from (going into) a pipe, so that the calling program may seek its input and output even if the program itself is part of a pipe.
NAME
blkio -- common code for get/put - blk/io
(only used internally)

SYNOPSIS
int unit;
short int ix,iy;
short int ixw,iyw;
int rw;        /* 01 or 02 for READ or WRITE
char*buf[BSIZE];

blkio("routine", unit, rw, ix, iy, ixw, iyw, buf, BSIZE);
NAME
blkxfer -- transfer a chunk of data -- internal use only

SYNOPSIS
int fd;        /* file descriptor
int rw;       /* read/write flag
char *buf;    /* buffer address
int cnt;      /* number bytes to xfer

blkxfer(fd, rw, buf, cnt);

returns:
   0 for operation successful
   1 for I/O error
**NAME**

getblk/putblk -- read/write an arbitrary chunk of a picture

**SYNOPSIS**

```c
int unit;           /* unit number
short int ix, iy;  /* upper left corner of chunk to be selected
short int ixw, iyw; /* x and y width of chunk
char buf[BUFSIZE]  /* buffer; BSIZE must be >= ixw*iyw

getblk(unit, ix, iy, ixw, iyw, buf, BSIZE);
putblk(unit, ix, iy, ixw, iyw, buf, BSIZE);
```

**returns:**

0 on successful operation
1 if buffer header bad
2 if buffer too small or lines out of range
3 if segment exceeds pix size
4 if file positioning error
5 if I/O error
NAME

col/putcol -- get or put a column of a pix.

SYNOPSIS

int unit; /* pds unit number
short int cnum; /* column number
char *buf[BSIZE]; /* buffer and size

col(unit,cnum,buf,BSIZE);
putcol(unit,cnum,buf,BSIZE);

returns:
0 for successful transfer
n>0 for any error (same as getblk/putblk)
NAME
getfmt -- read format info from header
putfmt -- initialize or change PDS header

SYNOPSIS
int unit;           /* unit number to read
short int fmt;     /* format of file
short int ix;      /* number pixels per line
short int iy;      /* number lines per pix
short int iz;      /* number bits per pixel
short int nc;      /* number channels

getfmt(unit, fmt, &ix, &iy, &iz, &nc)    /* note pointers used
putfmt(unit, fmt, ix, iy, iz, nc)       /* pointers NOT used

returns:
0) if all values returned ok
1) if unit # invalid or if unit doesn't correspond to an open file
2) if illegal argument values passed
NAME

ggetline/putline -- basic PDS I/O routines

SYNOPSIS

int unit;        /* unit #
short int lnum;  /* number of line to read or write
char buf[BSIZE]; /* buffer and size

ggetline(unit,lnum,buf,BSIZE); /* read line
putline(unit,lnum,buf,BSIZE);  /* write line

returns:
    0 if transfer ok;
n, n>0 if error:

    1) bad PDS header
    2) bad buffer (too small)
    3) can't position to line
    4) read or write I/O error

note:
The picture dimensions are specified by the pds header (stored internally), so the BSIZE parameter is redundant. However, it is included to allow additional error checking, to ensure that the programmer realizes how long the lines should be.
Small buffers are considered errors, while large ones are only filled as much as necessary. Only one line is transferred in any case.
NAME
    getttl -- get PDS title from header
    putttl -- initialize/modify PDS title

SYNOPSIS
    int unit;        /* unit #
    char tit[40];    /* 40 character array for title.

    getttl(unit, tit);
    putttl(unit, tit);

    returns:
    0 if successful
    1 if header bad
NAME
   itoa -- integer to ascii conversion (internal use)

SYNOPSIS
   int num;
   char buf[];
   itoa(buf,num);
NAME
pdsbok -- verify for buffer I/O -- internal use only

SYNOPSIS
int unit;        /* unit number
short int lnum;  /* requested line number
int bsize;       /* target buffer size (max)
char *routine;   /* name of calling routine

pdsbok(unit,lnum,bsize,routine)

1) see that line number is legal
2) see that buffer is large enough

Unit number must already been checked!!

returns 0 <=> everything ok.
NAME
pdsclose -- clean-up and close a PDS file

SYNOPSIS
int unit;
int access;
pdsclose(unit, access);

returns:
0 for successful close
n, n>0 for error as below:

1) unit # %d out of rang
2) unit # %d already closed
3) can't write header on unit %d
4) <any other PDS I/O errors>
NAME
   pdxexit -- close all PDS structures and exit.

SYNOPSIS
   pdxexit();

BUGS
   pdxexit should not be used if any pds files were opened with access = 4.
NAME
pdshok -- check that header is valid -- internal use only

SYNOPSIS
int unit;       /* unit number
char *routine;  /* name of calling routine

pdshok(unit, routine);

returns 0 <=> header ok.
NAME
   pdslpos -- position file for I/O of 1 line -- internal use only

SYNOPSIS
   pdslpos(unit,line)

BUGS
   all header checking must have already been done...
NAME

pdsopen -- open PDS descriptor file

SYNOPSIS

    int unit;        /* unit # to use
    char name[];     /* name of PDS file
    int access;      /* access mode to open file with
                     (same as for popnpds)

    pdsopen(unit, name, access)

DESCRIPTION

    This routine calls popnpds to do the actual file opening for it, so the access is
    exactly the same as that for popnpds. pdsopen also checks for correct unit
    numbers and allocates space for a new header each time it is called success-
    fully.

    returns:

    0 for successful open
    n, n>0 for each of the following errors:

    1) open called with bad unit #
    2) unit # already opened
    3) error returned from popnpds
    4) cannot get memory for header
NAME
pdsppos -- pixel position -- internal use only

SYNOPSIS
int unit;    /* unit number */
short int ix, iy;    /* position of pixel (ix, iy) in pix */

pdsppos(unit, ix, iy);

returns 0 <=> positioning successful
NAME
pdsuok -- check unit # is ok -- internal use only

SYNOPSIS
int unit;       /* unit number
char *routine;  /* name of calling routine

pdsuok(unit, routine);
returns 0 <=> unit ok.
NAME

vfyfmt -- verify PDS format name (internal use only)

SYNOPSIS

int unit;
char *buf;
vfyfmt(unit,buf);
NAME
Window Library Routines

AUTHORS
L. Dalle Ore and D. A. Kimber

DATE WRITTEN
5/80

SOURCE LANGUAGE
C

HOW TO WRITE A WINDOW PROGRAM
The first thing that should be included in any window program should be a #include <window.h> to set up all the necessary definitions for the window program and support routines. Then the default window size can be set by:

char *dsizex = "dxn";
char *dsizey = "dyn";

where dxn and dyn are the default size of the x and y coordinates respectively. Next, the help information should be put in helpstr[]. This ought to include the syntax for the input command line and a description of what the program does. If the input data is not in byte form, then the following steps need to be taken:

1) changing "#define pix char" and "#define nbits 8" in window.h to appropriate values for the new data type.

2) changing the mask of "&0377" to fit the new data wherever the mask occurs in the above programs.

This should be taken care of by introducing a "#define mask" in the programs to facilitate changing data types.

Now the main program, main(argc,argv) can begin. After defining all variables local to the window program, and before any executable statements, the call to initw(argc,argv,3,1); needs to be made to parse the input line so that any necessary files may be opened and required buffers allocated. This will also set up the pointers to any option strings. If OPTN is true, (non-zero) then an option was specified on the input line, and OPTN will be a pointer to the option string. Finally, the program itself may be written. When scanning the input in a window program the scanning should be done from left to right and from top to bottom, as this will result in the most efficient program. The x dimension should change faster than the y dimension. There are four main support functions for window programs; initw - parses the input line and opens files, getw - gets a window centered where specified, putp - outputs a pixel, and exitw - flushes all buffers and then exits. A complete listing of all the window support functions is in the appendix, along with complete descriptions of each.

The window library is linked using "-lpw".
To help clarify some of the rougher points here is an example window program that computes the local mean of the input window:

```c
#include <window.h>  /* window definitions */

char *dsizex = "2";  /* x dimension default size */
char *dsizey = "2";  /* y dimension default size */

char *helpstr[] = {
    "mean if=inf file=ofile [xs=xsize] [ys=y size]",  /* command syntax */
    "computes the local mean",  /* description of program */
    "default window size is 2x2",
    NULL};

main(argc, argv)
    int argc;
    char **argv;

{
    /* local variables */
    PIX cmean;
    float mean;
    extern struct wndw win;
    extern struct nwnd wout;
    int i,j,iw,jw;
    PIX **ibuf;

    initw(argc,argv,3,1);  /* parse input, open files, allocate buffers */

    /* perform the actual computation of the mean */
    for(j = 0; j < win.h.ysize ;j++)
        for(i = 0; i < win.h.xsize ;i++)
             ibuf = getw(&win,i,j);  /* getw gets a window, centered at i,j */
             mean = 0;
             for(jw=0; jw< win.ys; jw++)
                 for(iw=0; iw<win.xs; iw++)
                     mean = mean + (ibuf[jw][iw] & 0377);

             cmean = mean /(win.xs * win.ys);
             putp(&wout,&cmean,i,j);  /* putp writes an output pixel */
        }
    exitw();  /* exitw flushes buffers, closes files, and exits */
}
APPENDIX

For single input and single output (e.g. UNIX filter) window programs initw deals
with the opening of files. For multi input and multi output window programs the
user must write their own initialization routine.

Outline of available window support functions:

- clearw - clears an area of the buffer
- exitw - flushes all buffers still open, then exits
- fillbw - fills window for given file structure
- flushnw - flushes the output buffer
- getw - returns pointer to 2d array containing requested window
- getxw - reads a block into buffer from a file
- initw - parses input line, opens files and allocate buffers
- opnwin - open an input window file and allocate buffers
- opnwout - open an output window file and allocate buffers
- printw - prints a window for debugging purposes
- putp - puts a pixel on the output file
- putx - put an array on the output file
- setbnew - sets up the output buffer
- setbw - sets up the proper window
- set-edge - sets the picture edges to a specific value

NOTE

Integer arguments used in calling the above library routines must be declared
"short".
NAME

clearw

SYNTAX

clearw(w,x,y,nx,ny)

DESCRIPTION

clearw clears an area of the buffer. The array "clear" is used to skip over lines that are already clear.
Arguments are:

w address of the window structure
x x position of ulhc of area to be cleared
y y position of ulhc of area to be cleared
nx x dimension of area to be cleared
ny y dimension of area to be cleared
NAME
exitw

SYNTAX
exitw()

DESCRIPTION
exitw merely provides a nice exit by flushing all the buffers that are still open and then exiting.
NAME
fillbw

SYNTAX
fillbw(w,xc,yc)

DESCRIPTION
fillbw fills up a window for the given file structure.
Arguments are:

w   address of the window structure
xc  x position of the ulhc of window
yc  y position of the window

If the window is outside bounds the resulting elements will be cleared.
NAME
flushnw

SYNTAX
flushnw(w)

DESCRIPTION
flushnw flushes the output buffer.

Arguments are:
    w   address of the window structure

Will return a 0 on success, a 1 on error.
NAME
getw

SYNTAX
getw(w, xc, yc)

DESCRIPTION
getw returns a window from the input file. Arguments are:
w address of window structure (nwn)
xc x position of center pixel
yc y position of center pixel

Will return a pointer to a 2 dimensional array containing the requested window. Null returned on error.
NAME
getxw

SYNTAX
getxw(w,x,y,nx,ny,px,py)

DESCRIPTION
getxw will read a block into a specified position of the buffer from a given position in the file. Arguments are:
w address of the window structure
x,y x,y position on the file
nx,ny block size on file
px,py ulhc position in buffer
NAME
initw

SYNTAX
initw(argc, argv, inaccs, outacces)

DESCRIPTION
This is the initialization function for filter type window programs, i.e. those window programs with one input file and one output file.

Argc and argv are the same as those used by the main program. inacces and outacces are the access modes for the input and output files - those are used by popnpds to determine what to do for the case of pipes.

Access:

read = 0, write = 1
read and write = 2
read and seek = 3
NAME
opnwin

SYNTAX
opnwin(inacsrc, wsizex, wsizey, name)

DESCRIPTION
This function opens an input window file, allocates the window header and necessary input buffers and pointers. Arguments are:
inacsrc access mode for the input file- the same as usedby popnpds; 0=read, 2=read and write, 3=read and seek
wsizex the x dimension of the input window
wsizey the y dimension of the input window
name a pointer to the string containing the input file name

opnwin returns a pointer to the window header for a properly opened window file.
NAME
opnwout

SYNTAX
opnwout(outaccs, wsizex, wsizey, xs, ys, nchan, name)

DESCRIPTION
This routine opens an output window file and allocates the window header as well as the necessary input buffers and pointers. It returns a pointer to the window header for a properly opened window.
Arguments are:
outaccs - access mode with which the output file is to be opened, 1=write, 2=read/write 4=write and seek
wsizex - the x size of the window, this is used to allocate the output buffer correctly
wsizey - the y size of the window
xs - the x dimension of the output image
ys - the y dimension of the output image
nchan - the number of channels in the output image
name - a pointer to the string containing the name of the output file
NAME
printw

SYNTAX
printw(file,buf,wn)

DESCRIPTION
This function prints out a window for debugging purposes. Note: this function will work only for character data!
Arguments are:
  file    pointer to output file
  buf     pointer to pointers to window rows
  wd       window in use

To help explain the use of the functions opnwin and opnwout the following program is presented. It uses opnwin and opnwout instead of initw. For the calculation of the mean there is no advantage to using opnwin and opnwout, but for any programs requiring more than one input or output file they are a necessity. This program is strictly for use as an example.

#include <window.h>
/*
 * This program calculates the mean of the pixels in the window and outputs it to the center pixel position.
 * The difference between this program and "mean" is that this program does not use initw- instead it utilizes opnwin and opnwout to open the input and output window buffers and files.
 */

char *DSIZEX = "2";
char *DSIZEY = "2";

char *helpstr[] = {
  "mean if=ifile of=ofile [xs=xsize] [ys=y size]",
  "performs the mean value measurement",
  NULL
};

/* command line parameters array */
struct prs par[] = {
  "if=", 0,
  "of=", 0,
  "xs=", 0,
  "ys=", 0,
  "d", 0,
  ",", 0,
  "help", 0,
  "", 0,
  0, 0,
};
#define IVAL par[0].val
#define OVAL par[1].val
#define WSIZEX par[2].val
#define WSIZEY par[3].val
#define HELP par[8].val
#define ERRFLG par[7].val

char *DFLAG;  /* DFLAG must be specified so the window functions can use it for debugging if necessary */
struct nwnd *wout;
struct wndw *win;

main(argc, argv)
  int argc;
  char **argv;
{
  char nxspec = FALSE;  /* no x widow size flag */
  extern char _sobuf[ ];  /* used by setbuf to set up stack */
  PIX cmean;
  float mean;
  int i,j,iw,jw;
  PIX **ibuf;
  PIX **getw();

  setbuf(stdout, _sobuf);  /* set up stack for error recovery */

  mparse(argc, argv, par);  /* parse command line */

  /* check for command line options and errors */
  if(HELP){
    for(i=0;helpstr[i] != NULL; i++)
      fprintf(stdout,"%s,helpstr[i]);
    exit(1);
  }
  if(ERRFLG){
    fprintf(stderr,"unrecognized parameter %s,ERRFLG);
    exit(1);
  }

  /* calculate the window dimensions */
  if(!WSIZEX){
    nxspec = TRUE;
    if(!WSIZEY)
      WSIZE = DSIZEX;
    else
      WSIZE = WSIZEY;
  }
  if(!WSIZEY){
    if(nxspec}
WSIZEY = DSIZEY;
else
  WSIZEY = WSIZEX;
}

/* set the debug flag to "d" in the command line */
DFLAG = par[4].val;

/* open the input and output window files and headers */
win = opnwin(3,atoi(WSIZEX), atoi(WSIZEY), IVAL);
winout = opnwout(1, atoi(WSIZEX), atoi(WSIZEY),
  win->h.xsize, win->h.ysize, 1, OVAL);

/* calculate the mean */
for(j = 0; j < win->h.ysize ;j++)
  for (i = 0; i < win->h.xsize ; i++)
    if((ibuf = getw(win,i,j)))
      printf("error in getting window0);
      fflush();
      exit(1);
  mean = 0;
for(jw=0; jw<win->ys; jw++)
  for (iw=0; iw<win->xs; iw++)
    mean = mean + (float)(cfmt(ibuf[jw][iw]));
  cmean = iw = mean / (win->xs * win->ys);
  putp(wout,&cmean,i,j);
}
/* flush any data left in the output buffer to the output file */
flushnw(wout);

When using opnwin and opnwout the programmer must do several things that
weren't required of him with initw. The first of these is the specification of the
parameter array to be used by mparse. The flag DFLAG must also be defined for
use by the library routines for debugging purposes. If it is a null pointer, the no
default messages will be printed. Note that in the above program DFLAG is set to
par[4].val, the array element corresponding to the string "d". Thus, if d is typed
on the command line, the debug option will be selected. The user must also now
check for the help option and unrecognized parameter errors after return from
mparse. The programmer must also set up the default x and y window dimen-
sions for opnwin and opnwout.

One thing to be extra careful about is the fact that win and wout are now
pointers to window headers, NOT the names of the headers. This means that
items in the headers must be accessed via the win->xxx operator instead of the
win.xxx operator. (The compiler will not always catch it if these are switched).
Finally, the use of flushw for each output window assures that there is no data left in the buffers that hasn't been written to the corresponding output file.
NAME
putp

SYNTAX
putp(w,fp,xpos,ypos)

DESCRIPTION
This function puts a pixel on the output file.
Arguments are:

- w: address of the window structure (nwn)
- fp: pix pointer to pixel
- xpos: x position of pixel
- ypos: y position of pixel

Will return 0 on success, a 1 on error.
NAME
putx

SYNTAX
putx(w, x, y, nx, ny, buf)

DESCRIPTION
This routine was taken from trans.c and modified to write an arbitrary array on
the output file.
Arguments are:
w     address of the window structure
x     x position of ulhc
y     y position of ulhc
nx, ny number of lines to be written out
buf   buffer
NAME
set-edge

SYNTAX
set-edge(value)

DESCRIPTION
Set-edge will set all of the edges of picture to the value specified in (what else?) "value". This only works on byte pictures.

Arguments are:
  value    data value (int)
NAME
setbnw

SYNTAX
setbnw(w, name, accs, wxs, wys)

DESCRIPTION
This function sets up the output buffer.
Arguments are:

  w           address of the window structure
  name        string containing file name for window
  accs        mode of access (read or write)
  wxs         x size of window
  wys         y size of window
NAME
setbw

SYNTAX
setbw(w,name,acces,wxs,wys)

DESCRIPTION
This function sets up a proper window.
Arguments are:
w      address of the window structure
name    string containing file name for window
acces   mode of access (rd, read or wr, write)
wxs     x size of window
wys     y size of window