The DARPA/DMA Image Understanding Testbed PROGRAMMER'S MANUAL

Version 1.1

Technical Note 298

January 1984

Compiled by:

Kenneth I. Laws, Computer Scientist

Artificial Intelligence Center
Computer Science and Technology Division

SRI Project 1009

The work reported herein was supported by the Defense Advanced Research Projects Agency under Contract No. MDA903-79-C-0588.

333 Ravenswood Ave. • Menlo Park, CA 94025
(415) 326-6200 • TWX: 910-373-2046 • Telex: 334-486

The DARPA/DMA Image Understanding Testbed PROGRAMMER'S MANUAL

Version 1.1

Compiled by Kenneth I. Laws
Artificial Intelligence Center
SRI International

Foreword

The primary purpose of the Image Understanding (IU) Testbed is to provide a means for transferring technology from the DARPA-sponsored IU research program to DMA and other organizations in the defense community.

The approach taken to achieve this purpose has two components:

(1) The establishment of a uniform environment that will be as compatible as possible with the environments of research centers at universities participating in the IU program. Thus, organizations obtaining copies of the Testbed can receive a flow of new results derived from ongoing research.

(2) The acquisition, integration, testing, and evaluation of selected scene analysis techniques that represent mature examples of generic areas of research activity. These contributions from participants in the IU program will allow organizations with Testbed copies to immediately begin investigating potential applications of IU technology to problems in automated cartography and other areas of scene analysis.

The IU Testbed project was carried out under DARPA Contract No. MDA903-79-C-0588. The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the United States government.

This report provides UNIX-style programmer's reference documentation for IU Testbed software modules that are based on the UNIX system environment.

Andrew J. Hanson
Testbed Coordinator
Artificial Intelligence Center
SRI International

Abstract

This manual is a reference document for experienced programmers using the Image Understanding Testbed software system. It documents UNIX-based Testbed software modules and serves as a supplement to the UNIX Programmer's Manual. Included are descriptions of major Testbed contributions, utility programs, subroutine libraries, and data formats. All of the descriptions included here are also available interactively via the **man** command.

Section 1

Introduction

The Image Understanding Testbed is a system of hardware and software that is designed to facilitate the integration, testing, and evaluation of implemented research concepts in machine vision. The system was sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Defense Mapping Agency (DMA) and was developed at the Artificial Intelligence Center of SRI International.

This manual is intended primarily as a reference work for experienced programmers working in the Testbed UNIX environment. It contains a collection of **man** pages describing each major IU Testbed program and utility system, and parallels the "UNIX Programmer's Manual" which describes each UNIX system program and utility. Just as the source files for the **man** pages in the "UNIX Programmer's Manual" are in the directory tree */usr/man/man**, the source files for the **man** pages included here are in the directory tree */iu/tb/man/man**.

The documentation is divided up into several distinct parts. Part 1 describes main programs whose documentation is found in the *man1* directory with the extension ".1". Part 3 describes subroutines and utility systems whose documentation is found in the *man3* directory with the extension ".3" followed by a lower-case letter. These letters are used to distinguish different functional categories, and have the following meanings:

- **b** – Blklib object-oriented utility routines
- **c** – Command driver support routines, including CI and ICP
- **g** – Graphics routines
- **i** – Image file access routines
- **s** – Sublib routines comprising general utility functions
- **v** – Visionlib routines using both display and image access.

Part 5 contains file format information whose documentation is found in the *man5* directory with the extension ".5". Part 7 contains miscellaneous descriptive system information whose documentation is found in the *man7* directory with the extension ".7".

Each individual **man** entry can be viewed interactively on a terminal by invoking the command

man *command-name*

The Testbed **man**, **apropos**, **whatis**, and **whereis** commands have been modified to search both the UNIX and the Testbed *man* directory trees.

Introduction

A printed copy of any document included here can be generated by connecting to the appropriate */iu/tb/man* subdirectory, determining the full file name of the desired document, and invoking

vtroff *-man filename*

For an overview of the Testbed environment, including some elementary introductory material, see "The DARPA/DMA Image Understanding Testbed USER'S MANUAL."

Section 2

Summary index

The following is a chapter-by-chapter index of the Testbed **man** pages:

PART 1 – Main Programs

acronym (1)	- three-dimensional modeling and image imterpretation system
camdist (1)	- calculate camera models or distances to matched points
ccr (1)	- C program cross-referencer
clip (1)	- clip the gray levels in an image
convert (1)	- convert picture color coordinate systems
correlate (1)	- the Moravec stereo correlation package
describe (1)	- print parameters of a picture file
doc (1)	- create or edit manual entry
dpy (1)	- DPY-based image and graphics command driver
erase (1)	- clear a display window
ghough (1)	- Generalized Hough Transform Package
gmrsys (1)	- Grinnell checkout system
imgsys (1)	- image processing command driver
indent (1)	- indent and format a C program source
invert (1)	- reverse the vertical axis of a picture
line (1)	- Nevatia-Babu line finder
normalize (1)	- normalize the grayscale of an image
optronics (1)	- digitize film on the Optronics scanner
overlay (1)	- display an overlay on the Grinnell
phoenix (1)	- image segmentation by recursive region splitting
reduce (1)	- reduce an image by cropping or sampling
relax (1)	- CI-based relaxation package
sc (1)	- spreadsheet calculator
shapeup (1)	- reformat a picture file
show (1)	- display a picture on the Grinnell
showdtm (1)	- display representations of a digital terrain model
stereo (1)	- the Moravec multiple image correlator
view (1)	- formatted printout of a data file
whist (1)	- write a history message

PART 3 – Subroutines and Utility Libraries

arglib (3B)	- command-line and interactive argument query package
asklib (3S)	- ask user to type a value
blklib (3B)	- object-oriented structure manipulation
ci (3C)	- command interpreter
cmuimglib (3I)	- CMU image access package
cmunmelib (3I)	- CMU IMGNAME image name routines
del (3S)	- interrupt handling package
doclib (3S)	- routines for manipulating documentation records
dsplib (3G)	- device-independent display allocation

PART 1 — Main Programs

NAME

acronym – three-dimensional modeling and image imterpretation system

SYNOPSIS

acronym

DESCRIPTION

ACRONYM takes as input data a scene that has been reduced to a set of two-dimensional ribbons and a set of three-dimensional models. It searches the scene for instances of the models and identifies those which it finds. This is a rule-based system for interpreting preprocessed scenes in terms of classes of object models.

The ACRONYM modeling system involves a set of rules for each class of objects to be identified. Models are defined in terms of fundamental generalized cones, sweeping rules, and constraints; subclass conditions may be imposed to generate classes of similar objects.

The system is invoked by the command sequence:

cd /iu/tb/acronym/sys

acronym

The top-level commands are:

(load '../models/filename)
load up the desired model or models

(LOAD-IMAGE '/.../filename)
load up the desired image data filename

(PARSE)
begin passing the database over the image

(IDB) initialize the database

(G) display the existing models on the CRT.

R,E	roll and pitch
R,E	roll and pitch
R,E	roll and pitch
R,E	roll and pitch
R,E	roll and pitch
R,E	roll and pitch
R,E	roll and pitch
R,E	roll and pitch

(foobar maroo)
display the parsed image

(KWIT)
close the graphics device and exit. To close the graphics device alone, use **(g_close)**.

EXAMPLE

Commands issued in the following order should work.

acronym

```
        (load '../models/tbtank)
        (load ) (PARSE) (G) (+LSDFLKJDF) (IDB)  ; to start over again (KWIT)
```

FILES

/iu/acronym/sys – ACRONYM executable image and source files
/iu/acronym/compat – compatibility files for MACLISP to FRANZ LISP
/iu/acronym/doc – documentation for system builders /iu/acronym/graphics – graphics support routines
/iu/acronym/info – self-contained EMACS INFO system for ACRONYM
/iu/acronym/models – collection of models and images

BUGS

The original MACLISP function calls conflict with standard FRANZ LISP calls. The conflict is resolved by requiring that most original ACRONYM functions be invoked in upper case. FRANZ LISP functions are invoked in lower case. This makes the system drastically case-sensitive.

Invalid function calls drop into a break loop which cannot be exited using the customary ^Z. One must invoke **(reset)** to get back to the top level.

(G) has the default angle increments too large, so that almost any action moves the image off the screen unless the increments are halved several times.

HISTORY

08-aug-1983 hanson at SRI-IU
    Created.

NAME

camdist – calculate camera models or distances to matched points

SYNOPSIS

camdist [options]

DESCRIPTION

Camdist provides a facility for performing a generalized least-squares solution for the relative position and orientation angles between two cameras, given a series of points in the two camera views, and/or for calculating the distances to the points from such information. Wild points are automatically edited out, and the associated error propagation is done from the image plane points, through the camera model, to the distances.

Camdist is designed to be used in either a totally interactive mode or in a batch processing mode, depending on what options are given on the command line. The basic structure of the program is an infinite loop in which it asks for commands to execute. The commands available are:

c Solve for the camera model.

d Compute the distances to the points.

i Initialize parameters.

q Quit, exit the program.

The program always begins by executing the *i* command. If *camdist* is invoked without arguments, it proceeds to prompt the user for all needed information. Available options for automating it include (see *camdist(5)* for file formats):

-a [command-string]
If present, command-string is used to determine the first several commands which *camdist* responds to. If no command-string is given, the default command string *cdq* (calculate the camera model, compute the distances with it, then quit) is used.

-d distance-file
Output the distance results on the named file. (Default is to prompt the user for the file name.)

-i camera-input-file
Read the initial camera model values from the named file. (Default is to prompt the user for the camera model values.)

-m match-file
Read the matched point pairs from the named file. (Default is to prompt the user for the file name.)

-o camera-output-file
Output the calculated camera model on the named file. (Default is to prompt the user for the file name.)

-p [parameter-file]
If a file name is present, read various error parameters from the named file. If no file name is given, prompt the user for these parameters. (The default is to use the compiled-in values.)

-q Invoke quiet mode (turn off most of the informational type-out) while the camera solution is being sought.

FILES

/iu/tb/src/camdist contains the source files and the makefile.
/iu/tb/src/camdist/demo contains an example data set.
/iu/tb/src/camdist/doc contains an explanation of the variables used.
/iu/testbed/demo/camdist contains example demos of the program.

SEE ALSO

camdist(5)

DIAGNOSTICS

The program will complain if any of the necessary files do not exist. If the solution for a camera model proceeds badly (e.g. singular matrices), various informational messages will be printed.

BUGS

The camera calibrations procedure is somewhat unstable; small errors in the input data points can cause large changes in the camera model obtained.

HISTORY

18-Mar-83 Marsha Jo Hannah at SRI-IU
Changed name from *camrad* to *camdist*; installed latter on Testbed, after removing many bugs resulting from SAIL to C translation and changing the user interface to permit hands-off operation.

12-Aug-82 Bil Lewis at SRI-IU
Translated SAIL version of CAMRAD to C program *camrad.*

19-Jun-78 Bob Bolles at SRI-AI
Brought CAMRAD to 2060.

1972-1978 Don Gennery (DBG) at SU-AI
Created SAIL program CAMRAD.

NAME

ccr − C program cross-referencer

SYNOPSIS

ccr [pathname]

DESCRIPTION

Ccr uses the file pathname if given; otherwise reads from the standard input. It places a copy of the program with line numbers added on the standard output, followed by an alphabetical listing of variables and the line numbers on which they occurred. (Thus, **ccr** may be used as a filter.) Error messages and warnings always come to the terminal.

FILES

/iu/tb/src/ccr/ccr.c

DIAGNOSTICS

None.

NAME

clip – clip the gray levels in an image

SYNOPSIS

clip inpic [outpic] [**-below** minthresh [minval]] [**-above** maxthresh [maxval]] [**-else** [elseval]] [**-size** outbits] [**-output** outpic]

DESCRIPTION

Clip copies the monochrome input image to the output image with the specified clipping. If you omit the image names, you will be prompted for them. The default output name is the same as the input name, so be careful.

The output picture file name may be specified either positionally (following the input file name) or following a **-o** flag. It will have the same number of bits per pixel as the input image unless a **-s** flag is specified.

Image pixels below minthresh are set to minval; those above maxthresh are set to maxval. Default values are 0 for minthresh and the maximum possible output value for maxthresh; default values for minval and maxval are the corresponding thresholds. (You may want to set them to 0 instead.)

Pixel values between the two thresholds are ordinarily copied directly to the output image. You may replace them with a constant value by specifying the -e flag. The default value is the maximum possible output value. This is useful for creating a binary image.

Type **clip "??"** to print the synopsis line. (The quotation marks are necessary to get question marks past the shell parser.)

Arguments are parsed using the arglib package. Each flag introduces a new "command line" to be parsed. Special arglib flag values such as "??" or "?>" may be used with each of these clauses. (The code for this program is a good example of some very sophisticated parsing mechanisms.)

BUGS

Image data is treated as unsigned, but the command-line arguments are treated as signed. This could create problems on 32-bit images. Possibly signed pixels are better anyway, but it is difficult to establish default and legal thresholds for signed images.

Documentation associated with the input image is not copied.

SEE ALSO

normalize(1), piclib(3)

HISTORY

10-Oct-83 Laws at SRI-IU
Created.

NAME

convert – convert picture color coordinate systems

SYNOPSIS

convert {**HSI**|**YIQ**|**BW** inimg outimg

DESCRIPTION

Convert copies the RGB input image to the output image with the specified color coordinate conversion. If you omit the arguments, you will be prompted for them.

Image names are to be specified as templates with the feature omitted. Thus, /iu/tb/pic/chair/4.img refers to 4red.img, 4green.img, etc., in the directory /iu/tb/pic/chair. The program will fill in appropriate feature names for the input and output images.

Type

```
convert "??"
```

to print the synopsis line. (The quotation marks are necessary to get question marks past the shell parser.)

BUGS

This program demonstrates parsed name manipulation. It has an interesting user interface, but it is not necessarily better than the UNIX C-shell method of specifying picname/{red,green,blue}.img. It is convenient for specifying related image bands, but not for combining bands from various sources.

The program currently assumes that the output bands are to be put into the same directory (or generic image) as the input bands. This may be a poor assumption. More often the user wants to suppress the path and generic name.

There is no standard agreement on color coordinate formulas. The versions used here are the ones in /iu/tb/lib/visionlib. The hue formula produces output from 0 to 179 with red at both ends of the scale; achromatic pixels are assigned 255. Currently, I is stretched by 2 and Q by 4 before being shifted and clipped to fit an 8-bit range.

If you specify a generic name for the output image, the corresponding directory must exist. It would be nice if this program created the directory when necessary.

The old CMU image manipulation routines are used. The corresponding Testbed routines would be preferable.

Someday this routine should accept types of input other than RGB images.

Check the source code header for additional suggestions.

HISTORY

04-Oct-82 Laws at SRI-IU
Created from previous rgbtohsi, rgbtoyiq programs.

NAME

correlate – the Moravec stereo correlation package

SYNOPSIS

correlate [**-nqros#v#m** ffilename -ffilename]

DESCRIPTION

Correlate implements Hans Moravec's stereo correlation operator. "Interesting" points (corners with high contrast) are found in one image, and corresponding points are searched for in a second image.

-n Nodisplay. The default is to display the images, or as large a reduced version of the image as will fit, on the Grinnell display.

-q Quiet. Shuts off running commentary on the process.

-r Run Roberts' operator over each image and use the resulting images for the interest operator and correlator.

-o Run Sobel operator over each image and use the resulting images for the interest operator and correlator.

-s# Size of window used by correlator. Range is 1 through 10.

-v# Vertical hold. Maximum vertical deviation allowed for a match. If none is specified, looks for best match regardless of vertical stretch.

-m# Maxpoints. Maximum number of "interesting" points to be reported. Currently has an upper boundary of 99.

-t# Threshold. Threshold of "interestingness" for the interest operator. Range is 0 (default) through 255.

-i# Interestsize. Size of window for interest operator. Default is 4.

-ffilename
Name of image from which points are taken to be correlated. Must be specified exactly zero or two times. Note that this is the only specification that cannot be immediately followed by another specification with no separating blanks.

If the -f switch does not appear on the command line, the user is prompted for the image filename. If the -z switch does not appear, the user is prompted for correlator window size.

FILES

/iu/tb/src/stereo/*

EXAMPLE

correlate -s4
Will prompt user for filenames to be correlated using a 4 by 4 correlator window.

correlate -qnv40fphone/rgt.img -fphone/lft.img
Will correlate phone/rgt.img and phone/lft.img, accepting only points within 40 pixels of each other vertically in their respective images; will prompt user for window size; and will run quietly and without displaying the images.

DIAGNOSTICS

Prints error message and quits with a return value of 1 for files not found or unrecognized characters on command line.

BUGS

Check the source code header for suggestions.

HISTORY

08-Feb-83 Laws at SRI-IU

It is no longer necessary to specify full pathnames for images. The search path is taken from the PICPATH environment variable, and it defaults to ":/iu/tb/pic:/aux/tbpic".

15-Jul-81 Chuck Thorpe (cet) at Carnegie-Mellon University

Created.

NAME

describe – print parameters of a picture file

SYNOPSIS

describe filename

DESCRIPTION

Describe opens the header of a picture data file and prints its contents on the default output device. The information includes the header type, number of bits per pixel, the size of the picture, blocking structure of the file, and total number of bytes needed for the image data (excluding headers and trailers). Any printable documentation record is also printed out; see doclib(3).

FILES

/iu/tb/src/describe/*

SEE ALSO

show(1), whatis(1)

DIAGNOSTICS

The program will complain if an invalid picture file name is supplied.

BUGS

Describe should accept generic image names. It should also be extended to accept filenames, directories, etc.

Check the source code header for additional suggestions.

HISTORY

29-Nov-82 Laws at SRI-IU
Change the name from picparams to describe.

30-Aug-82 Hanson at SRI-IU
Created.

NAME

doc − create or edit manual entry

SYNOPSIS

doc [chapter] entryname ...

DESCRIPTION

Doc is a moderately high-level editor that allows you to conveniently create and edit manual entries without knowing any details of *nroff*(1). It allows the creation of moderately complex manual entries, and it is flexible enough to let you really *cut loose!*

The *chapter* parameter tells which chapter of the manual will contain this entry; it must begin with a number, but may end with a string (for example, *3m* for routines in the mathematics library). If you do not specify a *chapter*, 1 is assumed. The common chapters are: 1=programs, 3=subroutines, 5=conventions.

Entryname is the name of the program (etc.) you are documenting; if you give a list of entry names, they will all appear in the NAME section of the manual entry.

Doc works on a manual entry file named *entryname.chapter*, where *entryname* is just the first name in the list (if you specified several).

If no such file already exists, *doc* creates a new manual entry with all of the proper sections and no text. Then it allows you to type text into the first section. When you finish typing in all the sections desired for this manual entry, *doc* automatically executes *whist*(1) to add a standard format history message describing the creation of the entry.

If the manual entry file does exist, *doc* reads it in and makes the first section (after the NAME section) the current section. *Doc* will print a list of the sections present in the manual entry, and will print for you the last few lines of the current section. Frequently manual entries not created by *doc* will contain *nroff* commands that are not understood by the simple parser in *doc*. In such cases, *doc* will print a warning message, mark the command, and ignore it later. You may ignore these messages if you wish; *doc* will not understand these commands, but will preserve them in its output file unchanged. When you exit from *doc*, it will ask if you want to run *whist* to create a history message describing the changes you have made to the manual entry.

ENTERING TEXT

At any given instant, you will have a *current section* of the manual entry that is being edited and a *current line* after which you may insert additional text. When you type normal lines of text, they are added after the current line, possibly with some processing (for example, *doc* tries to ensure that each new sentence begins on a new line of text in the manual entry).

Doc can provide several formatting and other services for you, in response to *formatting commands* that you type. These are always entered by typing the *ESCAPE* key as the *first character* of a line of text. *Doc* will then accept a single letter that describes the formatting command you want.

When you have finished typing text for a section, type a line containing just a dot

(.). This section will exit from the current section and move on to the next section. If the next section already has some text, *doc* will print the last few lines of it for you; otherwise, just the name of the section will be printed. In either case, the current line will be the last line of the section, so that any text you type will be inserted at the end of the new section. (Note: This feature is exactly the same as ESCAPE-+, described below.)

FORMATTING COMMANDS

Doc allows you to specify certain *formatting commands* in addition to typing text. These commands are entered by typing *ESCAPE* as the first character on a line, then typing a character indicating which formatting command you desire. If you accidentally type an ESCAPE, just type another ESCAPE to get back into normal (text-entering) mode.

Some commands allow special formatting of the text in the manual entry.

b	*Break*. ESCAPE-b produces a "break" in the text that will force the next line of text to begin on a new line when the manual entry is printed. Normally all lines are filled; ESCAPE-b prevents line filling for the current line.
l	*Literal*. ESCAPE-l will put you into "literal mode." In this mode, line filling will not occur when this part of the manual entry is printed. Thus, the lines you type will be printed with the same spacing that you type. To leave this mode, type ESCAPE-r (*Reset*), described below.
t	*Table*. ESCAPE-t is used to create an entry in a table. The table will have the format of the table you are reading right now! To create such a table, for each entry you type: ESCAPE-t. *Doc* will type "Tbl" to show you that it understands what you want to do. You type the flag or entry name on the next line; it will appear at the normal left margin. All of the following text, as long as you are in table entry mode, will be indented somewhat farther. If the header is sufficiently short, the paragraph of text will start on the same line; otherwise, the text will start on the next line. To start the next table entry, just type ESCAPE-t again at the start of a text line; to get back into normal text mode, type ESCAPE-r (*Reset*), described below. When you switch to a new section, normal text-entry mode will also be resumed.
T	*Table Continue*. ESCAPE-T will allow you to specify the second and following lines of a table entry header which spans more than one text line. You type ESCAPE-t and the first line of the header, then ESCAPE-T and the second line, ESCAPE-T and the third, etc. When you finally type actual text, the usual table-entry mode rules apply (as described above for ESCAPE-t).
r	*Reset*. ESCAPE-r will exit from literal mode (ESCAPE-l) or table entry mode (ESCAPE-t), and resume normal text-entry mode. This command is automatically performed when you switch to a new section or exit from *doc*.
f	*Font*. ESCAPE-f will allow you to type text that will be printed in *italics* or **boldface**. You will be asked to type a letter indicating what font you wish (i or b). Boldface text only appears to be special when it is processed by an appropriate document production program; italics are printed with underlines on the Dover printer (see *cz*(1)) and on CRTs that can underline.

.,' *Nroff command.* ESCAPE-. or ESCAPE-' lets you type in an actual command for *nroff*(1) of the corresponding type. *Doc* will not understand this code; it will, however, faithfully remember to insert it in the manual entry at the appropriate place. This command is intended for people who think they are wizards and want to be proved wrong.

Some formatting commands tell you interesting things.

c *Context.* ESCAPE-c will print a few lines around the current line, just to remind you where you are inserting text in the manual entry. The format of these lines may be all wrong if they are part of a table entry; this command is just a reminder about where you are, not a simulator for *nroff*.

p *Print sections.* ESCAPE-p will print for you a list of the names of all the sections in this manual entry, indicating those that contain no text and which one is the current section.

? *HELP.* ESCAPE-? will print for you a list of all the formatting commands and a few tidbits of information to remind you how to enter text.

You can move around to different sections and create new sections.

+ *Next section.* ESCAPE-+ will exit text-entering mode on the current section and make the next section the current section. The current line will be the last line of the section. If the new section contains any text, the last few lines of it will be printed to remind you where the current line is. If the new section is empty, just its name will be printed. When you move off the end of the manual entry (i.e., do ESCAPE-+ when the current section is the last one), *doc* will exit (after performing *whist* and writing the manual entry, as previously described).

- *Previous section.* ESCAPE– is just the same as ESCAPE-+ (described above), but moves you to the preceding section instead of the following section. If you move off the end of the manual entry with ESCAPE–, a message will be printed and no action will occur.

s *Section.* ESCAPE-s will ask you for the name of a section of the entry that you wish to edit. You may type in lower-case; *doc* will automatically convert it to all capital letters. If the section already exists, it will be made the current section (just like ESCAPE-+); otherwise, it will be added to the manual entry (with no text) and then made the current section. If the current section has any text in it, the new section will be created ***after*** the current section; if, however, the current section is empty, the new section will be placed ***before*** the current section.

Some formatting commands execute interesting programs for you.

d,D *Display.* ESCAPE-d or ESCAPE-D will display the manual entry, so that you can see whether if it looks the way you want, by running the *man*(1) command. ESCAPE-d will display just the current section of the manual entry; ESCAPE-D will display the entire entry.

e,E *Edit.* ESCAPE-e or ESCAPE-E will allow you to edit the manual entry by creating a file, running an editor, and reading the file when you finish editing it. The *editor*(3) routine is used to call the editor, so the *EDITOR* environment parameter may specify which editor you like to use. If you have no EDITOR parameter in your environment, a default editor will be executed. The file you edit resembles the text you have typed, with *doc* commands imbedded in a special way (see INTERNAL FORMAT FILES, below).

! *Shell.* ESCAPE-! will fork off a shell for you. When you exit from the shell (by typing Control-D), you will be back in *doc*.

Finally, there are formatting commands to force *doc* to do special things with your manual entry file.

w *Write.* ESCAPE-w will cause *doc* to write your file for you. This is just like checkpointing; it gives you a good copy of the manual entry safe on the disk, in case you do something silly in the next few minutes.

q *Quit.* ESCAPE-q will cause *doc* to write your manual entry on the disk, run *whist* if needed, and exit. This is just like typing ESCAPE-+ in the last section of the manual entry, but you can do it any time.

PROTOTYPE FILES

Doc uses some special files, called *prototype files*, to tell it which sections are required for various kinds of manual entries. The prototype files are always called "xx," but *doc* will look for them in several places, as specified by the *MPATH* environment parameter. The prototype files may be in a subdirectory for the specific chapter of this manual entry (e.g. /usr/cmu/man/man1/xx), or in a single place for all manual entries to use (e.g. /usr/man/man0/xx). At CMU, each chapter of the manual has a prototype file that will be used by *doc* unless the user has his own prototypes.

A prototype file's first line begins with ".TH"; *doc* doesn't care what else is on this line. The remaining lines specify names of sections, either as

```
        .SH NAME
```

or

```
        .SH "NAME"
```

Doc ignores specifications for sections called NAME and HISTORY, since these will always be created; the other sections will be created with no text, and the first of them will be the section into which the user is initially typing text.

INTERNAL FORMAT FILES

Doc normally keeps the entire manual entry in a big data structure in core, writing it onto the disk only as needed. When you type ESCAPE-e or ESCAPE-E to edit the entry, a disk file is created. The editor is invoked to update this file; then the file is read back into *doc* to replace the text being edited.

The file sent into the editor does not contain *nroff* commands, which would be mysterious to anybody but an nroff wizard; instead, the special commands in the file are very similar to the formatting commands you type into *doc*. You can see what they are and understand what they do without too much difficulty; you can even add new commands (for example, if you decide that some word really ought to be italicized) or delete commands (if you change your mind again and decide it was right before).

Normal text lines are simply lines of text in the file you edit. Special commands always appear on lines by themselves, and always begin with the character '$.' Here are the special commands in these internal-format files:

$s	The following text line is the name of a new section, which starts here.
$b	Same as ESCAPE-b: insert a "break" here.
$l	Same as ESCAPE-l: enter literal mode.

$t Same as ESCAPE-t: enter table-entry mode, with the following text line as the table entry header.

$T Same as ESCAPE-T: the next line of text is the second (or later) line of a multi-line table entry header.

$~l, $~t
Similar to ESCAPE-r: indicate exit from literal mode to normal text mode, or from table-entry mode to normal text mode, respectively.

$fi, $fb
Same as ESCAPE-f-i and ESCAPE-f-b: the next line is to be printed in italics or boldface, respectively.

$., $' Same as ESCAPE-. and ESCAPE-': the next line is an nroff command of the corresponding kind.

$$ The next line looks like an nroff command, but is actually text.

$ The next line looks like a doc command (i.e., begins with $), but is actually text.

FILES

/tmp/doc.*	temporary files
/usr/cmu/man/man?/xx	CMU prototype files
/usr/man/man0/xx	Bell Labs prototype file.

ENVIRONMENT

MPATH
the list of places to look for prototype files used in creating new manual entries. If MPATH is missing from the environment, the places searched are the current directory, /usr/cmu/man, and /usr/man.

EDITOR
the name of the editor you like to use (see *editor(3)*).

SEE ALSO

man(1), nroff(1), whist(1), editor(3)
For a list of formatting macros, see man(7).

EXAMPLE

doc gorp
will allow you to enter (or edit) a manual entry for the program named *gorp*. The manual entry file will be called *gorp.1*.

doc 3s foo baz
will allow you to enter or edit a manual entry for the subroutines *foo* and *baz*, which appear in chapter "3s" of the UNIX manual. The manual entry file will be called *foo.3s*.

DIAGNOSTICS

If you interrupt *doc*, it asks you if you really meant to exit before writing the manual entry file. Remember that aborting *doc* will destroy its big in-core copy of the manual entry; you should first save the manual entry with ESCAPE-w.

If you try to edit an existing manual entry that contains random *nroff* commands, *doc* will complain:

```
            NROFF command not understood ...
```

You can ignore this message, if you want; it just means that *doc* will not be able to interpret that command. The command will, however, be put into the output file in the proper place without being changed.

BUGS

Doc should have some means for entering italicized and boldface table entry headers. Also, only a small subset of *nroff* is understood, making it difficult to use *doc* to edit manual entries that were created without *doc* (such as those from Bell Labs). It should be possible to nest literal mode and table entries inside of table entries, but it is not.

A "read <file>" command shold be available; it should not be necessary to invoke an editor.

The current method of branching on the first character of a line (using CBREAK) prevents deleting back to the start of a line.

Check the source code header for additional suggestions.

HISTORY

04-Feb-83 Laws at SRI-IU
Changed ioctl() calls to use CBREAK mode for the first character of each line – necessary for the <escape> machanism to work. Also added a manual() subroutine to replace calls to the CMU man program, which has evidently been hacked to recognize absolute pathnames and process them specially. Added erasure to the escape() backspace commands because our tty driver currently doesn't erase.

23-Nov-80 Steven Shafer (sas) at Carnegie-Mellon University
Modified for tty driver in Berkeley UNIX Distribution #4. Doc now asks you before deleting sections that contain no text.

15-Dec-79 Steven Shafer (sas) at Carnegie-Mellon University
Created.

NAME

dpy – DPY-based image and graphics command driver

SYNOPSIS

dpy

DESCRIPTION

Dpy is an ICP-based command driver. It was developed by Ron Cain and Todd Cushner for the SRI Printed Circuit Board Inspection project. It offers some multiwindow graphics capability, and it is integrated with the histogram manipulation code from the CMU Phoenix program.

The dpy and windowlib systems are characterized by multiple registered coordinate systems. A window is typically indexed from 0.0 to 1.0, although other scales may be defined. Routines to convert to and from absolute screen coordinates are available.

The main difference between this and a CI command interpreter is that ICP permits direct invocation of subroutines (i.e., no interface routine is normally needed) and the routines are able to return values.

The following menu documentation was provided with the program. For a demonstration, connect to /iu/tb/src/dpy/demo and "<" in the show or windowtst command file.

DPY Window System Routine Guide

```
DPY *dpyinit(display_mode)
   int display_mode;

   display_mode =
      W_BW: black and white mode.
      W_RGB: rgb (color) mode.


infodpy(dpy)
   DPY *dpy;

   dpy = pointer to some dpy.


DPY *show(dpy,ptr_pic)
   DPY *dpy;
   IMGDPY *ptr_pic;

   dpy = pointer to a dpy to show image in.
   ptr_pic = pointer to an image file discriptor record.


DPY *window(dpy,xlow,ylow,xlen,ylen)
   DPY *dpy;
   double xlow,ylow,xlen,ylen;
```

```
    dpy = pointer to a dpy to make window in.
    xlow,ylow = lower-left coordinates of new window.
    xlen,ylen = length of window (dpy's coordinate system).


setdpy(dpy,option,value)
    DPY *dpy;
    int option,value;

    dpy = pointer to the dpy to be changed.
    option =          W_OVERLAY: set the overlay colors.
      W_BOX: set the box the window mode.
    value = overlay color.


amove(dpy, xpos, ypos)
    DPY *dpy;
    double xpos,ypos;

    dpy = pointer to window to make absolute move in.
    xpos,ypos = absolute location to move to.


rmove(dpy,xinc,yinc)
    DPY *dpy;
    double xinc,yinc;

    dpy = pointer to window to make relative move in.
    xinc,yinc = delta x and y change for new location.


adraw(dpy,overlay,size,xpos,ypos)
    DPY *dpy;
    int overlay,size;
    double xpos,ypos;

    dpy = pointer to draw line in.
    overlay = color to draw overlay line with.
    size = size of the line.
    xpos,ypos = coordinates of the endpoint of the line.


rdraw(dpy,overlay,size,xinc,yinc)
    DPY *dpy;
    int overlay,size;
    double xinc,yinc;

    dpy = pointer to window to draw in.
    overlay = overlay color for the line.
    size = size of the line.
    xinc,yinc = delta x and y for the line endpoint.
```

```
text(dpy,overlay,size,direction,text_string)
  DPY *dpy;
  int overlay,size,direction;
  char *text_string;

  dpy = pointer to a window to write text in.
  overlay = color of overlay for the text.
  size = size of the font to be used.
  direction = direction the text is to be drawn.
  text_string = pointer to the character string to be used.


clear(dpy,mask)
  DPY *dpy;
  int mask;

  dpy = pointer to window to clear.
  mask = bit string indicating what to clear:
    W_WHITE, W_RED, W_GREEN, W_BLUE, W_DISPLAY.


cursor(dpy,xpos,ypos)
  DPY *dpy;
  int *xpos,*ypos;

  dpy = pointer to a window to use.
  xpos,ypos = pointers to the ending location of the cursor.


IMGDPY *openimg(image_type,image_name)
  int image_type;
  char *image_name;

  image_type =
    W_BW: black and white image file to be opened.
    W_RGB: rgb image file to be opened.
  image_name = full UNIX path name for image file.
```

FILES

/iu/tb/src/dpy/*
/iu/tb/src/dpy/pcblib/*
/iu/tb/lib/visionlib/windowlib/*

SEE ALSO

imgsys(1), icp(3), windowlib(3)

DIAGNOSTICS

The blklib routines underlying the picture access routines use the printerr error reporting package. The pcblib and windowlib routines do not.

BUGS

Much of the documentation is in windowlib(3).

The DPY and windowlib systems have not been strongly integrated with the Testbed or with the blklib system. Eventually dpy and imgsys should be combined (unless the DPY concept is completely replaced by a new Testbed graphics system). Another ICP system for graphic display is also being developed by Jim Herson at SRI.

The DPY structure grew in parallel with the Testbed image access structures. Either DPY or a similar datatype (e.g., a PORT or VIEW) should be reworked to include IMGs, PICs, FRMs, DSPs, etc.

This system has not been used extensively. Both it and the ICP driver need further work.

The fitpic() routine should expand images as well as shrink them.

Check the source code headers for additional suggestions. See also the ICP documentation.

HISTORY

05-Feb-83 Laws at SRI-IU
Created.

NAME

erase – clear a display window

SYNOPSIS

erase [minx miny maxx maxy]

DESCRIPTION

Clears (i.e., sets to zero) the Grinnell display. If limiting coordinates are specified, only that window is cleared. Minx and miny default to 0.0, maxx and maxy default to the maximum screen dimension.

For a usage reminder, type

```
erase "??"
```

To get prompting, type

```
erase "?>".
```

FILES

/iu/tb/src/erase/erase.c

SEE ALSO

overlay(1), show(1)

DIAGNOSTICS

The routine currently will not allow you to specify a window that extends outside the display window. This can be easily changed if device-independent window specification proves desirable.

BUGS

Currently the program does not read back the display registers to determine which memory channels are in use. Sets the display to monochrome and only erases the specified window in memory plane 0. Perhaps a memory plane specification should be added as an argument, or at least -rgb and -all flags.

The window specification could be made to accept a -n specification in the manner of show(1). This would permit specification by corner and size instead of by two corners.

Check the source code header for additional suggestions.

HISTORY

29-Nov-82 Laws at SRI-IU
Changed the name from clearpic to erase.

21-Sep-82 Laws at SRI-IU
Created.

NAME

ghough – Generalized Hough Transform Package

SYNOPSIS

ghough

DESCRIPTION

GHOUGH is a program for detecting instances of a given shape within an image. The instances may be displaced, rescaled, or rotated relative to the original shape template. Shape instances are detected by computing a "generalized Hough transform" of the image edge elements. Each edge element votes for all those instances of the shape that could contain it. The votes are tallied and the best supported instances are reported as likely matches. This local voting method permits partial or occluded shapes to be detected, as well as instances of any size or orientation.

Top-level commands:

draw [column row angle radius]
: Draws the template with the specified position and size. You will be asked for any parameters not specified. Use erase if you want to clear the screen before drawing another template.

erase Erases any overlay planes, but does not erase the image. Erase is often used just before the draw routine. It is not needed before a search because the search automatically erases overlays before beginning.

open [image]
: Displays the image and makes it available for tracing or searching. If no image is specified, you will be asked for one.

print Prints the numeric values in the template table. This command is generally only used for debugging.

quit n Quit the current level of the CI driver. At the top level, this command will close the current picture file and leave GHOUGH. The argument n may be provided to quit more than one level; n == 1 by default. Specify n == -1 or some large number to abort all levels of the driver and exit the program.

```
search [(mincolumn,minrow) (maxcolumn,maxrow)
    deltacolumn deltarow
    minangle maxangle deltaangle
    minradius maxradius deltaradius
    edgethresh contrast dump]
```

Searches the window from (mincolumn,minrow) to (maxcolumn,maxrow) for the template shape. Note the parentheses, which are required. You may omit arguments by coding successive commas, e.g.

```
search ,,1,1,20,30,2,6,8,1,120,black,no.
```

to have the routine ask for interactive specification of the window. The edge search and the Hough search for shape centers are both registered with this search window. You may specify the spatial resolution of the search using deltacolumn and deltarow. Setting these to two is equivalent to analyzing only every other column and row of the image.

Minangle, maxangle, and deltaangle determine the angular search range and resolution. Rotation angle is always specified clockwise from the orientation at which the template was traced.

Minradius, maxradius, and deltaradius determine the shape sizes to be found. The numbers are specified in pixel units, but decimal fractions are permitted.

The edge threshold determines edge strength required before a pixel can vote for candidate shapes. A value of 120 will pick up most strong edges, but you should specify a smaller number to locate objects that match their backgrounds. The amount of time required for a Hough analysis is linearly proportional to the number of edges that are above the threshold.

Contrast must be specified as black, white, or mixed. Black is used to find dark objects and white to find light ones. Specify mixed if you do not know the object/background contrast or if the contrast can vary from one part of the shape to another.

Dump is to be specified as yes or no. (It often makes more sense to omit it from the command line and answer the question when it arises.) Specify yes only if you want to dump out some of the Hough accumulator planes as images so that you may study them using other programs. If you specify yes, the program will ask a number of additional questions.

The search routine will locate edges, build the Hough accumulator, find potential shape matches (local maxima in the accumulator), and then enter a deeper level of command driver. The help files for that level will be made available to you at that time. Basically, you will have a menu of commands for studying the list of potential matches and for displaying them on the screen. When you quit that level, the dump query will be executed and then the search will be finished.

Use "verbose = yes" to enable additional printout during searches.

select [color]

Erases the current overlay and selects a new overlay color. The current choices are

1 - Blue
2 - Green
3 - Red
4 - White.

If you omit the color number, you will be asked for one.

sobel [size]

Permits you to read the Sobel gradient and gradient angle at selected points in the image. The keyboard cursor routine is invoked so that you may select the image points.

You may also specify a Sobel mask size, for which the mask used will be 3*size by 3*size. Each size by size element of the mask is averaged to form the 3 x 3 array used in computing the Sobel gradient.

template [file]
: Reads a template file. If no template is specified, you will be asked for one.

trace
: Permits you to build a new template file. The keyboard cursor routine is invoked so that you may indicate the center of the figure and any number of points around its boundary. (Type "help cursor" for more information on controlling the cursor.) Be sure to specify the silhouette points in clockwise order (beginning anywhere). The underlying image is not used by the program; in particular, no use is made of image gradient at the points you select. (The edge direction at each template point is assumed to be the direction between its two neighboring points.)

verbose [= yes | no]
: Controls whether the search command will print statistics about accumulator size, edges found, execution times, etc.

During the search command, another level of the CI driver is invoked. This level permits the user to control display of the identified match points. Use "symbol = yes" to turn on full-shape display, "no" to display only center points (except for the "display" command, which always draws the full shape).

The available commands are:

above [threshold]
: Displays all Hough matches above the specified accumulator count threshold. If the threshold is not specified, it will be requested.

display [match]
: Display the specified match using the full template shape. If no match number is given, you will be asked for one.

erase
: Erases any overlay planes, but does not erase the image. Erase is often used just before the draw routine. It is not needed before a search because the search automatically erases overlays before beginning.

first [N]
: Display the first N matches in the sorted list of Hough accumulator maxima. If N is not specified, 20 matches or the maximum number found will be used. Note that the matches displayed are not necessarily better than those omitted.

level [N]
: Display a specified level of the match list. [If N is omitted, the top level is shown.] The Hough count of the best match defines the first level. Each succeeding level has one less Hough count, even if no matches were found at that level.

quit n
: Quit the current level of the CI driver. At the top level, this command will close the current picture file and leave DSPSYS. The argument n may be provided to quit more than one level; n == 1 by default. Specify n == -1 or some large number to abort all levels of the driver and exit the program.

save [filename]
: Save the current overlay as an image file. If no filename is given, you will be asked for one.

score [filename]
: Take the specified file as a ground truth file. [If no name is given, you will be asked for one.] The file should contain one ASCII line for each known target specifying the column, row, orientation, and radius. This list will be checked against the Hough match list to determine how many of the known targets were found. The hits will be displayed.

select [color]
: Erases the current overlay and selects a new overlay color. The current choices are

  1 - Blue
  2 - Green
  3 - Red
  4 - White.

  If you omit the color number, you will be asked for one.

symbol [= yes | no]
: Controls whether the various display commands (other than "display") will use the full template shape for each match. The initial setting is "no," which causes only the center points to be displayed.

top [N]
: Display the top N levels of the match list. [If N is omitted, only the top level is shown.] The Hough count of the best match defines the first level. Each succeeding level has one less Hough count, even if there are no matches at that level.

FILES

/iu/tb/src/ghough/ghough.c

SEE ALSO

ci(3)

DIAGNOSTICS

GHOUGH requires the display and will fail if it cannot obtain one.

BUGS

If templates with very dense points are used, the interpoint angles will nearly all be multiples of 45 or even 90 degrees, and performance will be greatly degraded.

Check the source code header for additional suggestions.

HISTORY

09-Dec-82 Laws at SRI-IU
Created.

NAME

gmrsys – Grinnell checkout system

SYNOPSIS

gmrsys

DESCRIPTION

This is a very primitive CI driver, written before we understood either CI or the Grinnell. It contains an assortment of canned routines to test various features of the CMU Grinnell software. The best way to use this program is to study the examples and then write your own or extend the existing ones.

To get a list of commands, type "*". Particularly useful is the "hexloop" command for sending raw hex codes to the display. You may also use the debug command to examine the hex codes being sent by the other commands.

FILES

/iu/tb/lib/src/gmrsys/*

SEE ALSO

dpy(1), imgsys(1), dsplib(3)

BUGS

Several of the routines should be rewritten to accept command-line or interactive arguments.

Check the source code header for additional suggestions.

HISTORY

05-Feb-83 Laws at SRI-IU
Created.

NAME

imgsys — image processing command driver

SYNOPSIS

imgsys

DESCRIPTION

Imgsys is an ICP-based image processing command driver. It originally provides a basic menu sufficient to display subwindows of picture files in windows on the default display device. Type "?" for a list of commands. You may extend this capability by attaching additional menus.

Most of the blklib routines available from imgsys will accept a variable number of arguments and will ask for any required arguments that you omit. A few blklib structures (with the value NULL) are initially provided to prevent the routines from asking for their values.

With all of the menus attached, imgsys becomes essentially an interpreter for the C language. It lacks structured datatypes and dynamic subroutine definition, however.

The main difference between this and a CI command interpreter is that ICP permits direct invocation of subroutines (i.e., no interface routine is normally needed), and the routines are able to return values.

FILES

/iu/tb/src/imgsys/*

SEE ALSO

dpy(1), relax(1), blklib(3), dsplib(3), icp(3), imglib(3), piclib(3)

DIAGNOSTICS

The blklib routines use the printerr error reporting package.

BUGS

This system has not been used extensively. Both it and the ICP driver need further work.

It has certainly not been established yet that a C interpreter is the best user interface for an image processing workstation. A CI driver is able to provide a more customized interface, although it has difficulty manipulating multiple images or display windows.

The system has several limitations that result from the underlying ICP driver. Adding an extended menu, for instance, does not eliminate identical routines in the basic menu; this lack prevents the abbreviation facility from functioning correctly. The use of a variable name that is a leading substring of a routine name (e.g., pic, or even p) can also cause problems.

The extended menus should be provided with argument lists and defaults. Currently you are expected to provide all the arguments.

Imgsys needs commands for cursor motion and pixel query, subwindow extraction, and overlay management. It also needs commands for display memory flicker, scroll, CRT on/off, save and restore state, etc. Higher level functions such as transposing, thresholding, and mosaicing should also be developed.

Imgsys needs a better ICP query facility for requesting the names of all open images, pictures, windows, etc.

Imgsys also needs color image display, which should be implemented by extending the "IMG" datatype to include any pixel-oriented data source or sink.

Interactive graphics commands are needed. For a fancy system, it should be possible to specify the pen color using color names instead of RGB or IHS coordinates. (See the Color Naming System in *IEEE Computer Graphics*, May 1982.)

There should be a path variable so that imgsys knows where to look for picture files. Picture-name templates should be supported as they are in Phoenix. [This support has now been provided by the PICPATH environment variable and the ! escape mechanism.]

Perhaps imgsys should open and close the Grinnell on program startup just to let the user know if the display is unavailable. (This procedure makes less sense if we are hooked up to multiple displays.)

The quit or exit command should take an optional argument. With no argument, it should clear the display on exit so that other users will know it is free.

At the University of Maryland, a history is maintained in the header of each picture file. The history of the parent image is copied, then the command line of the current main program is appended. (At the University of Maryland, pictures are never altered, only transformed during copy.) If this is a useful idea, we should develop some analogous process (such as the graphics trail file) within imgsys.

Closing a PIC or other structure currently leaves a dangling pointer. One could solve this by "pic0 = close(pic0)", but this approach is rather clumsy. A previous driver had defined pic0 to be a pointer to the PIC (using the undocumented ICP := command), but this definition proved to be too obscure and too likely to pass bad pointers; it also required all blklib routines to expect such arguments. The only other cure would be to modify ICP to intercept closing commands and null the associated pointers.

A mechanism should permit structures created by function calls in a statement but not assigned to any variable to be "closed" automatically after the statement is finished. Thus,

```
openimg (picture "bw.img" 0),,(window 0 0 63 63)
```

would create and then close a picture, a window, and an image. A garbage collection system might be implemented for this purpose.

ICP itself should be available from IMGSYS so that we can add some kind of "push_level" command to demo scripts. We could also use a simple echo and halt facility.

Check the source code header for additional suggestions. See also the ICP documentation.

HISTORY

05-Feb-83 Laws at SRI-IU
Created.

NAME

indent − indent and format a C program source

SYNOPSIS

indent ifile [ofile] [args]

DESCRIPTION

The arguments that can be specified follows. They may appear before or after the file names.

ifile	Input file specification.
ofile	Output file specification. If omitted, then the indented formatted file will be written back into the input file, and there will be a "back-up" copy of ifile written in the current directory. For an ifile named "/blah/blah/file", the backup file will be named ".Bfile". (It will only be listed when the '-a' argument is specified in ls.) If ofile is specified, indent checks to make sure it is different from ifile.
-lnnn	Maximum length of an output line. The default is 75.
-cnnn	The column in which comments will start. The default is 33.
-cdnnn	The column in which comments on declarations will start. The default is for these comments to start in the same column as other comments.
-innn	The number of spaces for one indentation level. The default is 4.
-dj,-ndj	-dj will cause declarations to be left justified. -ndj will cause them to be indented the same as code. The default is -ndj.
-v,-nv	-v turns on "verbose" mode, -nv turns it off. When in verbose mode, indent will report when it splits one line of input into two or more lines of output, and it will give some size statistics at completion. The default is -nv.
-bc,-nbc	If -bc is specified, then a newline will be forced after each comma in a declaration. -nbc will turn off this option. The default is -bc.
-dnnn	This option controls the placement of comments which are not to the right of code. Specifying -d2 means that such comments will be placed two indentation levels to the left of code. The default -d0 lines up these comments with the code. See the section on comment indentation below.

-br,-bl Specifying -bl will cause complex statements to be lined up like this:

```
if (...)
{
    code
}
```

Specifying -br (the default) will make them look like this:

```
if (...) {
    code
}
```

You may set up your own 'profile' of defaults to indent by creating the file '/usr/your-name/.indent.pro' (where your-name is your login name) and including whatever switches you like. If indent is run and a profile file exists, then it is read to set up the program's defaults. Switches on the command line, though, will always over-ride profile switches. The profile file must be a single line of not more than 127 characters. The switches should be seperated on the line by spaces or tabs. Indent is intended primarily as a C program indenter. Specifically, indent will:

> indent code lines

> align comments

> insert spaces around operators where necessary

> break up declaration lists as in "int a,b,c;".

It will not break up long statements to make them fit within the maximum line length, but it will flag lines that are too long. Lines will be broken so that each statement starts a new line, and braces will appear alone on a line. (See the -br option to inhibit this.) Also, an attempt is made to line up identifiers in declarations.

Multi-line expressions
Indent will not break up complicated expressions that extend over multiple lines, but it will usually correctly indent such expressions which have already been broken up. Such an expression might end up looking like this:

```
x =
    (
      (Arbitrary parenthesized expression)
      +
      (
        (Parenthesized expression)
        *
        (Parenthesized expression)
      )
    );
```

Comments
Indent recognizes four kinds of comments. They are straight text, "box" comments, UNIX-style comments, and comments that should be passed thru unchanged. The action taken with these various types is as follows:

"Box" comments: The DSG documentation standards specify that boxes will be placed around section headers. Indent assumes that any comment with a dash immediately after the start of comment (i.e. "/*-") is such a box. Each line of such a comment will be left unchanged, except that the first non-blank character of each successive line will be lined up with the beginning slash of the first line. Box comments will be indented (see below).

Unix-style comments: This is the type of section header which is used extensively in the UNIX system source. If the start of comment ('/*') appears on a line by itself, indent assumes that it is a UNIX-style comment. These will be treated similarly to box comments, except the first non-blank character on each line will be lined up with the '*' of the '/*'.

Unchanged comments: Any comment which starts in column 1 will be left completely unchanged. This is intended primarily for documentation header pages. The check for unchanged comments is made before the check for UNIX-style comments.

Straight text: All other comments are treated as straight text. Indent will fit as many words (separated by blanks, tabs, or newlines) on a line as possible. Straight text comments will be indented.

Comment indentation Box, UNIX-style, and straight text comments may be indented. If a comment is on a line with code it will be started in the "comment column", which is set by the -cnnn command line parameter. Otherwise, the comment will be started at nnn indentation levels less than where code is currently being placed, where nnn is specified by the -dnnn command line parameter. (Indented comments will never be placed in column 1.) If the code on a line extends past the comment column, the comment will be moved to the next line.

DIAGNOSTICS

Diagnostic error messsages, mostly to tell that a text line has been broken or is too long for the output line, will be printed on the controlling tty.

FILES

/usr/your-name/.indent.pro – profile file

BUGS

Doesn't know how to format "long" declarations.

NAME

invert – reverse the vertical axis of a picture

SYNOPSIS

invert inpic outpic

DESCRIPTION

Invert copies the input picture file to the output file with the row order reversed. If you omit the input and output picture names, you will be prompted for them.

Type

```
invert "??"
```

to print the synopsis line. (The quotation marks are necessary to get question marks past the shell parser.)

BUGS

The default is to ask whether you want the output picture to have the same name as the input picture. Be careful.

Invert was written as a demonstration program for programmers wanting an example of piclib usage. It uses pixel-by-pixel data copying, which is the most general form but hardly the fastest. It should be rewritten to use row-by-row copying.

Check the source code header for additional suggestions.

HISTORY

14-Oct-82 Laws at SRI-IU
Created.

NAME

line – Nevatia-Babu line finder

SYNOPSIS

cd /iu/usc/tst
../bin/convolve
../bin/thrin
../bin/psmaker
../bin/linkseg
../bin/segdisp

DESCRIPTION

This is a primitive C-coded version of the Nevatia-Babu line finder system which was written originally in SAIL. It does not include the supersegment and super-antiparallel segment capabilities. Each routine prompts the user for any required arguments.

Further documentation is available in the Hughes Aircraft document *C Version of the Nevatia-Babu Line Finder*.

FILES

/iu/usc/bin, /iu/usc/tst, /iu/usc/*.

BUGS

Most parameters are entered interactively, with little guidance regarding defaults and meaning of the inquiry. The system should be expanded to supply reasonable defaults, to be more user-friendly, and to include interactive display capabilities.

HISTORY

08-Aug-83 Hanson at SRI-IU
Created.

NAME

normalize – normalize the grayscale of an image

SYNOPSIS

normalize [**-b**<number>] [**-t**<fraction>] inimage [outimage]

DESCRIPTION

Normalize examines the histogram of *inimage* and constructs a new image on *outimage* that represents the same image with as few bits per pixel as possible without distorting the histogram.

The *-t* (tail) flag specifies the maximum fraction of pixels that may be mapped into black or white. By default, it is 0.05.

The *-b* (bits) flag may be used to specify the number of bits per pixel in the result. By default, this is derived by squashing the histogram's tails just short of violating the tail constraint and noting the number of gray levels that are still to be spanned.

If *outimage* is not specified, then the result replaces *inimage* (same name, different i-node).

BUGS

The extreme output values are reserved for the clipped tail pixels. If no clipping is needed, these bins are used as part of the ordinary histogram. As a result, the semantics of the black and white output gray levels depend on the input image.

Normalize might be able to use the histogram maniputlation routines in vision-lib, particularly hcutoff().

Check the source code header for additional suggestions.

SEE ALSO

clip(1)

HISTORY

26-Jul-82 Laws at SRI-IU
Extended the routine to include histogram stretching. Added the bug note on tail bin semantics.

07-Jan-80 David Smith (drs) at Carnegie-Mellon University
Created.

NAME

optronics – digitize film on the Optronics scanner

SYNOPSIS

optronics

DESCRIPTION

This is a flexible user interface to the Optronics film digitizing system. It uses the CI driver *CI*(1) as an interactive command system. Upon entering the program, the system prompts the user for all the essential parameters needed to start a scan.

COORDINATE SYSTEM

The coordinate system on the scanner is represented by the following diagram:

```
0-----------> Y <drum axis = direction of carriage motion>
|
|
|
X
<direction of drum rotation>
```

Coordinates are in centimeters.

TRANSMISSION MODE

The allowed values of the **pixel** parameter are

12.5   25   50   100   200   400

in microns for transmission mode (transparent film). *BOTH* thumbwheels, top and bottom, on the scanner must be set to correspond with the panel switch setting and with the pixel setting value in this program. The fiber optics slider inside the right lower cabinet door must be set on TRANSMITTANCE.

The 12.5 micron setting is not recommended for color images.

REFLECTANCE MODE

Only the 100 micron setting (100R on the bottom thumbwheel) may be used for reflectance mode (opaque film). The fiber optics slider inside the right lower cabinet door must be set on REFLECTANCE.

COLOR DIGITIZING

Color images may be scanned in three passes by changing the color filter wheel inside the right lower cabinet door from the transparent **O** setting to **R**, **G**, and **B** in sequence.

SCANNER PANEL CONTROLS

The user should be familiar with the following controls:

ON/OFF switch
: flip switch inside the left lower cabinet door

Density switch
: rotary switch with values 3.0, 2.0, 1.0, 0.5 inside left lower cabinet door. Usual setting is 2.0. The density D is given by the formula:

  log10(1/transmittance)
  : This means that

    D=0.5 –> values 0–255 map to 31.6% to 100% transmittance

    D=1.0 –> values 0–255 map to 10.0% to 100% transmittance

    D=2.0 –> values 0–255 map to 1.0% to 100% transmittance

    D=3.0 –> values 0–255 map to 0.1% to 100% transmittance

LOG/LIN switch
: on main console, selects the indicated data transform method. Usual setting is LOG.

RUN switch
: on main console, starts the drum turning so that data acquisition can begin.

raster size switch
: rotary switch on main console, selects the size of pixel to be digitized, 12.5, 25, 50, 100, 200 or 400 microns. This switch must agree with the thumbwheel settings. **NEVER CHANGE THIS SWITCH WHILE THE DRUM IS TURNING.**

COMMANDS

The following commands are available to the user once the scanner is initialized:

run Begin digitizing data from the scanner using the current parameters.

put *filename*
: Save the current digitized image in the named file.

params
: Print the current parameters

range *x1 y1 x2 y2*
: Set the range of the scan in centimeters.

density *value*
: Set the scanning density. Valid values are 0.5, 1.0, 2.0, 3.0.

pixel *value*
: Set pixel size - must correspond to the thumbwheel and raster switch settings. Valid values are 12.5, 25, 50, 100, 200 and 400 microns.

positive/negative
: Positive film (complement the film density values), or negative film (use raw film density values).

log/linear

Logarithmic or linear conversion of actual film density to output value.

title Enter a descriptive picture title.

erase Erase the display

show Display the current digitized image on the Grinnell

control
Send direct control characters to the Optronics - for debugging.

move *y*
Move to the designated drum carriage position.

quit Exit from the system.

FILES

/iu/tb/src/optronics/optronics.c — source file
/tmp/optro.pic — temporary picture file

DIAGNOSTICS

If the drum is not turning because the **run** switch has not been activated, the system will complain and ask the user to turn the system on.

BUGS

A **range** command must be entered after a **pixel** command in order to update all the variable tables.

If the x origin is given the illegal value "0.0", the resulting data will be invalid.

HISTORY

09-Aug-83 Hanson at SRI-IU
Created.

NAME

overlay – display an overlay on the Grinnell

SYNOPSIS

overlay [picname color mincol minrow]

DESCRIPTION

The specified file is overlayed on Grinnell memory 0 and the device is set up to display this memory plane.

Color may currently be red, green, blue, or white.

Mincol and minrow are the lower-left pixel of the display window. If not specified, the image will be centered. The image is clipped to fit the display memory.

For a command synopsis, type

```
overlay "??".
```

To get prompting for arguments, type

```
overlay "?>".
```

FILES

/iu/tb/src/overlay/overlay.c

SEE ALSO

erase(1), show(1)

DIAGNOSTICS

The program currently objects if no pixel of the picture will be visible on the screen. (This could easily be disabled.)

Prints a warning if the image must be clipped.

BUGS

Currently displays only monochrome images and a single overlay.

Check the source code header for additional suggestions.

HISTORY

24-Sep-82 Laws at SRI-IU
Created.

NAME

phoenix – image segmentation by recursive region splitting

SYNOPSIS

phoenix inimage **-o** outimage **-f** feat1 [feat2 ...] [**-i** file | **-I** file] [**-e**] [**-s**] [**-O** file **-r** reg# **-R** reg#]

DESCRIPTION

PHOENIX is an image segmentation program based on the work of Ronald Ohlander at Carnegie-Mellon University. Previous versions – one called Kiwi and another named Moose – were designed and implemented by Steven A. Shafer, who also designed this program. A few modifications have also been made during integration with the Image Understanding Testbed.

Segmentation is accomplished by a process of recursive region splitting. At each stage, an analysis is performed on histograms for all available image features. As a result of the analysis, a feature is selected and the region being processed is thresholded. The resulting new regions are then determined by means of a connected region extraction algorithm developed by Shafer for Kiwi.

The program takes as input a digital image consisting of one or more features, e.g., red, green, blue, hue, saturation, intensity, etc. Each feature is represented by a picture file. The output of the program is a 16-bit region map, a Testbed picture file in which the pixel values are region numbers.

The program expects input image files to be named according to the conventions adopted by the CMU vision group and documented in imgname(5) and the CMU IUS white paper "Image File Naming Conventions." The feature field should be omitted from *inimage*, which will thus stand for a set of images, each representing a different feature. The program will determine the features to be used by looking at the arguments of the **-f** flag. The program expects to find a picture file for each feature specified. See the EXAMPLE section.

Command line options:

-i file Read commands from "file" before accepting commands from the terminal. The command file is interpreted by ci(3); any command that can be issued to PHOENIX interactively can be issued from a command file.

-I file Read commands from "file" and then exit without accepting commands from the terminal.

-e Echo commands read from a file. Normally commands are not echoed as they are interpreted from a command file.

-s Rather than take commands from the terminal, issue a single "segment" command and then exit. This is not likely to be useful unless some commands are read from a command file; e.g., you will have to set "flags=ABC" to get an uninterrupted segmentation, and you may want to set "flags=q" to squelch tty output.

-O file This and the -r and -R flags allow one to open an existing region map (from a previous run of PHOENIX) for further segmentation. Normally the program creates a new output region map. The -r and -R flags are used to specify which region should be considered for further segmentation and which is the highest numbered region in the map.

-r reg#
: Specify the initial region to be segmented when updating a previously created region map. (See -O above.)

-R reg#
: Specify the largest region number in the region map to be updated. (See -O above.)

COMMANDS

PHOENIX uses the ci(3) command interpreter to provide the user with a sophisticated interface for interactive execution and control of the segmentation process. The commands that are currently available are described briefly below. The interactive help facility of the PHOENIX program gives a more detailed description of the commands.

abort	Terminate segmentation of current region.
checkpoint	Save the current state of the segmentation.
clear	Remove all regions from a queue
describe	Display information about a data structure.
display	Display data structures on the Grinnell.
dqueue	Remove regions from a specified queue.
exit	Terminate PHOENIX.
history	Describe the ancestors of a region.
list	Display the elements of a specified queue.
prune	Excise a portion of the segmentation tree.
queue	Add regions to a queue.
restore	Restore a checkpoint file.
retry	Reexecute a previous phase of segmentation.
rundisplay	Initiate continuous Grinnell display.
segment	Proceed with the next phase of segmentation.
transfer	Move regions from one queue to the other.

FLAGS

Global Segmentation Flags

Internal, user-accessible flags are used to control program execution. Each is represented by an alphabetic character, and the list of those that are set is stored in a variable called 'flags'. Just typing 'flags' at the PHOENIX command prompt will cause a list of the currently SET flags to be displayed. The user can modify the list by typing 'flags=<mod string>' where <mod string> is defined as follows:

<mod string> = <modifier> | <mod string><modifier>
<modifier> = [+|-]<letter> | [+|-]<asterisk>
<letter> = any upper or lower case alphabetic char
<asterisk> = the character '*'

For example, to cause an initiated segmentation to continue until the next level in the segmentation tree is reached, and then to pause at that point, one should type 'flags=ABP'. This will cause the A, B, and P flags to be set and will have no effect on other flags. Note that '+' can be omitted when there is no preceding '-' and that the effect of either sign continues until another sign is encountered. The '*' is an abbreviation for all flags. Thus, if you wanted to have only the A and B flags set, you could type 'flags=-*+AB', which will first cause all flags to be unset and then cause the A and B flags to be set.

The following flags are currently defined:

A	Continue to the next phase (default).
B	Continue to the fetch phase.
C	Continue to the next level.
D	Enable debug printout of storage management messages.
G	Enable debug printout of subroutine entry, exit.
H	Enable interval phase cutpoint selection display.
P	Pause rather than stop on scheduled interrupt.
d	Describe fetched region or intervals.
g	Order regions globally by area.
m	Manual decision on whether to segment each region.
o	Order regions within each depth by area (default).
q	Execute quietly, i.e., without tty output.
v	Autoverbose mode (default).

Local Phase Flags

It is also possible to have flags set or unset during particular phases of segmentation. The format of the command is: during <phase> <mod string>, where <phase> is the name of the phase of segmentaton during which the global flags are to be modified as indicated by <mod string>.

VARIABLES

Interval Set Variables (Default value)

absarea	Minimum allowed interval area (10).
absmin	Eliminate valleys higher than this multiple of the lowest valley (10).
absscore	Minimum allowed interval set score (700).
height	All interval peaks must be greater than this percentage of the second highest apex (20).
intsmax	Maximum number of intervals in an interval set (2).
isetsmax	Maximum number of interval sets that will be selected (3).
maxmin	Intervals with lower max-to-min ratios are eliminated (160 [1.6-to-1]).
relarea	Minimum allowed percentage of total histogram area (2).

relscore	Minimum allowed percentage of highest set score (60).

Misc. Modifiable Variables (Default value)

depth	Maximum depth of the segmentation tree (maximum integer).
noise	Patches with smaller areas are considered noise and are eliminated (10).
retain	Maximum percentage of region area that is allowed to be noise (in selected regions) (20).
splitmin	Minimum sized region to be automatically split (40).
tolerance	Tolerance for polygon fitting (0.1).

Read-only Variables

features	Features being used in segmentation.
images	Images being used in segmentation.
lastphase	Last segmentation phase completed.
nextphase	Next segmentation phase.
phases	List of all segmentation phases.
regions	Number and range of existing regions.
time	CPU and real times for all phases.

EXAMPLE

In the following example, PHOENIX is executed to segment an image, presumably of a car, that is defined by three files that contain the data for the red, green, and blue features. The "(phoenix)" prompt is displayed, and the program waits for a command from the user. The user then sets some of the global flags and initiates the segmentation.

```
% phoenix car/1.img -o car/1map.img -f red green blue

PHOENIX (Version 1)            Carnegie-Mellon University

(phoenix) flags=BP
flags          ABPov
(phoenix) segment
```

FILES

/iu/tb/src/phoenix/*/*
/tmp/phoenix.??? - Temporary threshold image file

/iu/tb/src/phoenix/demo/strict.cmd - Strict (cautious) heuristics
/iu/tb/src/phoenix/demo/moderate.cmd - Moderate heuristics
/iu/tb/src/phoenix/demo/mild.cmd - Mild (permissive) heuristics

SEE ALSO

ci(3), cmuimglib(3), names(5), CMU IUS white paper #4, "Image File Naming Conventions," by Steven J. Clark.

This offer arrived with the code: "If you need assistance in determining appropriate values for the global variables, contact Steven Shafer at Carnegie-Mellon University." His current net address is Steve.Shafer@CMU-10A.

BUGS

A directory named "/tmp" is expected to exist. Temporary threshold images are written in this directory.

The D, G, v, and o flags need to be documented on-line.

The queue commands permit multiple occurrences of a region on one queue.

Source images used for a checkpoint cannot be renamed or the restore command will fail.

The heuristic screening routine added by SRI should allow a "quit" reply to escape from the display sequence.

The "delete" interrupt needs to be fixed.

The "bug" command does not work on the Testbed system.

The "release" command does not complain if you don't have the display.

The "display" command kills the composite overlay normally maintained by rundisplay. The scheduler should maintain the status of this region map so that it can be restored if it has been erased.

PHOENIX is strongly tied to the image naming convention. It will not accept a single color image band for monochrome segmenation. Neither will it accept a pair of color bands--it insists on having all three.

There is no help file for hsmooth.

Check the source code headers and the source directory for additional suggestions.

HISTORY

09-Dec-82 Laws at SRI-IU
Modified this file to conform to the Testbed version.

07-Sep-81 Duane Williams (dtw) at Carnegie-Mellon University
Now describes version 1.

08-May-81 Duane Williams (dtw) at Carnegie-Mellon University
Created.

NAME

reduce – reduce an image by cropping or sampling

SYNOPSIS

reduce [**-amoslu**] [-#] [**-h**#] [**-v**#] [**-c**#[:#]] [**-r**#[:#]] [**-q**] InImage OutImage

DESCRIPTION

Reduce processes *InImage* to produce *OutImage*. It can reduce the size of the image by cropping and/or reduction over a window. The flags are as follows:

a	(default) Reduce by averaging over a reduction window. (A, m, o, s, l, and u flags are mutually exclusive.)
m	Take median over window.
o	Take mode over window
s	Sample image.
l	Take the lowest value in each window.
u	Take the uppermost value in each window.
#	(a bare number) Set the horizontal and vertical sizes of the reduction window *(hfactor* and *vfactor)* simultaneously to #.
h#	Set *hfactor (horizontal reduction window size) to #*.
v#	Set *vfactor* to #.
c#	Set the starting column of the cropping window. The ending column is the last column in the InImage.
c#:#	Set both starting and ending columns of cropping window.
r#	Set the starting row of the cropping window. The ending row is the last row in the InImage.
r#:#	Set both starting and ending rows of the cropping window.
q	Run in quiet mode (default is noisy).

FILES

/iu/tb/src/reduce/reduce.c

SEE ALSO

normalize(1)

EXAMPLE

```
reduce -c370:569 -r100:299 /usr/vision/pix/dc1419m4.img dc.img
        crops a 200 x 200 window out of the source image.
```

BUGS

Don't try this on images with more than 10 bits unless you sample. (Only 1024 cells of histogram storage are allocated.)

Check the source code header for additional suggestions.

HISTORY

26-Jul-82 Kenneth I. Laws at SRI-IU
Added documentation on the l and u options.

15-Jan-80 David Smith (drs) at Carnegie-Mellon University
Created.

NAME

relax — CI-based relaxation package

SYNOPSIS

relax

DESCRIPTION

Invokes a relaxation system based on the CI command interpreter. Type * for a list of commands or help * for a list of help files. The commands are:

clear Clear the display.

defcom compatfile relaxtype nlabels [neighborfile]
: Defcom is an interactive program used to install hand-computed compatibility coefficients for each neighbor of a point. The arguments specify the file to which the coefficients are to be written, whether a Hummel-Zucker-Rosenfeld ("h") or Peleg ("p") relaxation is desired, the number of labels in the relaxation process, and, optionally, a file that specifies a nonstandard neighborhood.

defnbr neighborfile ncols nrows
: Defnbr is an interactive program used to define a nonstandard neighborhood for each point. The "neighborfile" argument specifies the file to which the neighborhood definition is to be written. The number of columns and rows that can contain the neighborhood must also be specified. The program will ask which is to be considered the center point and whether each neighbor is to be considered in the neighborhood.

hcompat prbfile compatfile [neighborfile]
: Computes the Hummel-Zucker-Rosenfeld compatibility coefficients for the probability file (default prb.img) and stores them in a compatibility file (compat.dat). You may specify a neighborhood file as created by defnbr.

  The hcompat program must have been compiled previously; see setup. If you would like to specify the compatibility coefficients by hand, see defcom.

hrelax prbfile compatfile [neighborfile]
: Performs one Hummel-Zucker-Rosenfeld relaxation operation on a probability file (default prb.img) using compatibility coefficients in compatfile (compat.dat). You may specify a neighborhood file as created by defnbr.

  The hrelax program must have been compiled previously; see setup.

imgprb inimg nlabels minval maxval
: Convert an image to probability format with the specified number of labels. The image will be clipped (stretched) using minval and maxval as the outer gray-level limits: omit these or specify -1 if the full input range is to be used. The file "prb.img" will be produced as output. At present it is just a floating-point data file rather than a true picture file.

insert picname mincol minrow
: Insert the picture into the display at the specified lower left pixel position.

pcompat prbfile compatfile [neighborfile]
: Computes the Peleg compatibility coefficients for the probability file

(default prb.img) and stores them in a compatibility file (compat.dat). You may specify a neighborhood file as created by defnbr.

The pcompat program must have been compiled previously; see setup. If you would like to specify the compatibility coefficients by hand, see defcom.

prbimg outname minval maxval
: Convert a probability file to image format with the specified output range. Omit minval and maxval or specify -1 to use the full (8-bit) dynamic range of the output image. The input file is assumed to be "prb.img".

prelax prbfile compatfile [n neighborfile] [s nsets size1 ...]
: Performs one Peleg relaxation operation on a probability file (default prb.img) using compatibility coefficients in compatfile (compat.dat). You may specify a neighborhood file as created by defnbr; precede it with the letter n. You may also specify the grouping of label values into sets.

  The prelax program must have been compiled previously; see setup.

quit n
: Quit the current level of the CI driver. At the top level, this will leave RELAX. The argument n may be provided to quit more than one level. Specify -1 or some large number to abort all levels of the driver and exit the program.

setup {h|p} nlabels [ncols nrows]
: Setup is used to create programs for relaxation operations on images. Both a compatibility operator and a relaxation operator will be compiled. Either Hummel-Zucker-Rosenfeld (h) or Peleg (p) relaxation formulas may be chosen; the corresponding programs will be hcompat and hrelax or pcompat and prelax. The default is "h". You may also specify the number of class labels (default 2) to be used and the size of the relaxation neighborhood (default 3 x 3).

EXAMPLE

To compile a Hummel-Zucker-Rosenfeld relaxation operator, compute the probability image, and compile the compatibility coefficients, use

```
setup h
imgprb bw.img
hcompat
```

To run the relaxation and display the result, use one or more steps of

```
hrelax
prbimg output.img
insert output.img
```

FILES

```
/iu/tb/src/relax/*
/iu/tb/src/relax/src/*
prb.img
compat.dat
```

SEE ALSO

imgsys(1), ci(3)

DIAGNOSTICS

Insert complains if it cannot open the Grinnell. You may still run the relaxation commands.

BUGS

Each step executes a separate program, so there is very little communication between the steps. Each "insert" command, for instance, opens and then closes the display.

The temporary files are always named prb.img and compat.dat.

Check the source code header and source directory for additional suggestions.

HISTORY

09-Dec-82 Laws at SRI-IU
    Included the help file for each command.

21-Sep-82 Laws at SRI-IU
    Created.

NAME

sc − spreadsheet calculator

SYNOPSIS

sc [file]

DESCRIPTION

Sc is a calculator that is based on rectangular tables, in much the same style as VisiCalc or T/Maker. When it is invoked, it presents you with an empty table organized as rows and columns of entries. Each entry may have a label string and an expression associated with it. The expression may be a constant or it may compute something based on other entries.

When *sc* is running, the screen is divided into three regions: the top line is for entering commands, the second line is for messages from *sc*, and the rest form a window looking at the table. *Sc* has two cursors: an entry cursor (indicated by a '<' on the screen) and a character cursor (indicated by the terminal's hardware cursor). The entry and character cursors are often the same. They will differ when a long command is being typed in the top line.

The following single control character commands are recognized no matter where the character cursor is:

^N	Move the entry cursor to the next row.
^P	Move the entry cursor to the previous row.
^F	Move the entry cursor forward one column.
^B	Move the entry cursor backward one column.
^C	Exit from *sc*.
^G	Abort the current long command.
^H	Backspace one character.
^L	Redraw the screen.
^J	Create a new row immediatly following the current row. It is initialized to be a copy of the current row, with all variable references moved down one row. If an expression is to be duplicated with ^J, the moving down of a variable reference may be avoided by using the "fixed" operator.
^V	Type, in the long command line, the name of the entry being pointed at by the entry cursor. This command is used when typing in expressions to refer to entries in the table.
^U*n*	Sets the numeric argument for the following command to *n*. Commands like ^F and ^B use the numeric argument as the number of times to

perform the operation. If you aren't entering a long command, the ^U is unnecessary; repetition count arguments may be entered as just a string of digits.

The following commands are only valid when the character and entry cursors are the same; that is, when no long command is being entered. Most of them introduce a new long command.

= Prompts for an expression that will be evaluated dynamically to produce a value for the entry pointed at by the entry cursor. This may be used in conjunction with ^V to make the value of one entry be dependent on that of another.

? Types a brief helpful message.

Enter a label for the current entry.

e Edit the value associated with the current entry. This is identical to '=' except that the command line starts out containing the old value or expression associated with the entry.

< Associate a string with this entry that will be flushed left against the left edge of the entry.

> Associates a string with this entry that will be flushed right against the right edge of the entry.

g Get a new database from a named file.

p Put the current database onto a named file.

w Write a listing of the current database in a form that matches its appearance on the screen. This differs from the "put" command in that "put"s files are intended to be reloaded with "get", whereas "write" produces a file for people to look at.

f Sets the output format to be used for printing the numbers in each entry in the current column. Type in two numbers that will be the width in characters of a column and the number of digits that will follow the decimal point.

r Create a new row by moving the row containing the entry cursor and all following rows down one. The new row will be empty.

c Create a new column by moving the column containing the entry cursor and all following rows right one. The new column will be empty.

d	Delete this row.
D	Delete this column.

Expressions that are used with the '=' and 'e' commands have a fairly conventional syntax. Terms may be variable names (from the ^V command), parenthesized expressions, negated terms, and constants. The +/ term sums values in rectangular regions of the table (the notation +/ is reminiscent of APL's additive reduction.) Terms may be combined using many binary operators. Their precedences (from highest to lowest) are: *,/; +,-; <,=,>,<=,>=; &; |; ?.

e+e	Addition.
e-e	Subtraction.
e*e	Multtplication.
e/e	Division.
+/v:v	Sum all valid (nonblank) entries in the region whose two corners are defined by the two variable (entry) names given.
e?e:e	Conditional: If the first expression is true, then the value of the second is returned; otherwise, the value of the third is.
<,=,>,<=,>=	Relationals: true if the indicated relation holds.
&,|	Boolean connectives.

FILES

/iu/tb/src/sc/*.c
/iu/tb/src/sc/demo/expense.sc – a sample expense report.

SEE ALSO

apl(1), bc(1), dc(1), the VisiCalc or T/Maker manuals.

BUGS

There should be a */ operator.

Expression reevaluation is done in the same top-to-bottom, left-to-right manner as that used in other spreadsheet calculators. A proper following of the dependency graph with (perhaps) recourse to relaxation should be implemented.

Check the source code header for additional suggestions.

NAME

shapeup – reformat a picture file

SYNOPSIS

shapeup [inpic outpic hdrtype scantype blocktype [blockcols blockrows]]

DESCRIPTION

Shapeup reformats a picture file. It is typically used to convert a foreign picture format to Testbed format. If you do not ask to be queried, only the input and output file names will be requested; output will have the Testbed format and default scan and blocking factors.

Output is with a Testbed header, LRBTSCAN, and 32 x 32 blocking unless these options are overridden on the command line. You may specify "?" for an argument that is to be queried, or "?>" to force querying on all succeeding arguments. (The quotation marks are necessary to get question marks past the shell parser.)

To convert to the SRIHDR format, it is easiest to type

```
shapeup inpic outpic SRIHDR
```

To get an image with no header, type

```
shapeup "?>"
```

and answer the interactive queries.

You may type

```
shapeup "??"
```

to print the synopsis line.

BUGS

The default is to ask whether you want the output picture to have the same name as the input picture. Be careful.

Shapeup tends toward verbosity. Perhaps it needs a -q "quiet" flag. UNIX philosophy would tend toward a -v "verbose" flag.

It should be possible to override the normal parsing of the input file so that one could specify the blocking and other parameters instead of taking them from the (possibly inaccurate) header. At present, this can only be done by compiling a modified shapeup that will open the image in raw mode.

Additional options should be implemented. Pixel type needs to be provided as soon as we support more than UNSIGNED pixels. Color image copying might be desirable, in which case the method of specifying the image name will have to be improved.

Shapeup currently ignores any proplist or comment information in the image header. This should be changed. (We will first have to get proplists working for testbed images.)

Check the source code header for additional suggestions.

HISTORY

16-Oct-82 Laws at SRI-IU
Changed default scan direction name to LRBTSCAN.

10-Sep-82 Laws at SRI-IU
Created.

NAME

show – display a picture on the Grinnell

SYNOPSIS

show [picname ...] [-**insert**] [-**from** mincol minrow [maxcol maxrow]]
[-to mincol minrow [maxcol maxrow]] [-**n** ncols nrows]

DESCRIPTION

The specified picture is displayed on the Grinnell. Previous contents of the Grinnell memory are erased unless the -i flag is specified. The display configuration will be altered as necessary to display the new image properly. The color type will be determined from the number of picname arguments:

BW One file name (prior to the flags) will cause a monochrome image corresponding to the named picture file to be displayed.

STEREO
Two file names will cause an anaglyphic stereo picture to be displayed; the left image (displayed in red) corresponds to the first file name and the right image (displayed in green) corresponds to the second.

RGB Three file names will cause a color picture to be displayed. The first file name is displayed in red, the second file name in green, and the third file name in blue.

You may specify, in any order, the source window (-f), the destination window (-t), or the window size (-n). If the -f and -t specifications are omitted, the corresponding windows will be centered. If the window size is never specified, the full picture size will be used. If it is redundantly specified, each specification must be smaller than the previous one; the last one is then used.

The source window must currently be contained entirely within the source image. The destination window may be partially outside the physical display memory limits; the image will be clipped to fit. If it is completely outside, a warning message will be printed and no further action will be taken.

For a command synopsis, type

```
show "??"
```

You may also type

```
"?>"
```

after any flag to be prompted for the flag arguments.

FILES

/iu/tb/src/show/show.c

SEE ALSO

erase(1), overlay(1)

DIAGNOSTICS

The program will complain if the picture file or files specified do not exist or if the Grinnell is allocated to another user and cannot be opened.

Prints a warning if the image must be clipped or if no pixels will overlap the display memory.

BUGS

STEREO images are displayed rather slowly. Greater speed has been incorporated into the new CMU frame package, but is not currently available on the Testbed.

If you use show to browse through large images, be sure to specify the viewing window size. Show centers the image instead of registering it at a corner of the screen; this can cause confusion if the image is larger than the screen.

Check the source code header for additional suggestions.

HISTORY

01-Jan-83 Laws at SRI-IU
Changed the argument parsing mechanism to be cleaner and more intelligent. Allowed display completely outside the display memory bounds.

29-Nov-82 Laws at SRI-IU
Changed the name from insertpic to show. Added the -i flag.

28-Oct-82 Laws at SRI-IU
Added color capability.

04-Oct-82 Laws at SRI-IU
Added flag arguments and default completion of window specifications.

21-Sep-82 Laws at SRI-IU
Created.

NAME

showdtm – display representations of a digital terrain model

SYNOPSIS

showdtm [dtm-name [command-string [parameter-file-name]]]

DESCRIPTION

The named digital terrain model is displayed on the Grinnell in the mode specified. Previous contents of the Grinnell memory are erased.

This is an interactive program intended to give the user great flexibility in displaying a digital terrain model and producing either a perspective grid plot or a perspective range image of a portion of a model. When invoked with no arguments, showdtm will prompt for the name of a terrain model (an image in testbed format), then wait for commands. If the name of an image file is specified, the program will open that, then wait for commands. If an initial command string is specified, the program will execute each of those commands, then wait for more. If the name of a parameter file is specified, the program will use those parameters to initialize the many internal variables (see examples on /iu/testbed/demo/mtpark). Commands that will be of interest to the casual user (in approximate order of usefulness) are:

Z Clear the Grinnell.

S Show (display) the specified part of the digital terrain model. (The default is the entire terrain model, or its lower left corner if it is larger than the Grinnell screen; these parameters can be changed through the I command, below.) For 8-bit images, this is equivalent to just displaying the image. For 16-bit DTMs, only the low-order 8 bits are shown, giving the effect of contouring the data at an interval of 256 units.

P Depending on the mode selected (see D below), do either a perspective grid plot or a perspective range image of the specified portion of the terrain model. (The default area and view parameters are tuned to the one example we have at this time; these parameters can be changed through the I command, below.) The perspective grid plot (done on the red frame of the Grinnell) is approximately what one would see if someone were projecting a grid of light lines onto a solid model; hidden line elimination occurs. The range image is similar, except the facets of the model are "painted" with the range from the focal point, which is expressed as a 16-bit quantity and displayed in the green and blue frames. (Note that it is advisable not to do a range image until you are satisfied with the grid plot--range images are SLOW.)

1 Switch the Grinnell to the DTM display, on the black-and-white frame.

2 Switch the Grinnell to the grid/range plot(s), on the color frame.

? Type out a menu of commands.

Q Quit and exit the program.

I Initialize all parameters for plotting. This walks the user through all of the initializing procedures for specifying which part of the DTM to display, how frequently to display lines, which part of the DTM to plot (the Testbed keyboard cursor commands are used to point to the corners of the area on the displayed DTM), and the camera position and orientation (again, the keyboard is used to move the focal point and stare point). If you didn't get it right, do another I--most of the values will be unchanged, and can be kept by simply replying <return> to the prompt. (Do a P command to see the results.)

D Prompt the user to set the grid/range mode for the plot (takes effect on the next P command).

FILES

/iu/tb/src/showdtm contains the source files and the makefile.
/iu/tb/pic/mtpark contains an example data set.
/iu/testbed/demo/mtpark contains two example demos of the program.

DIAGNOSTICS

The program will complain if the DTM file specified does not exist or if the Grinnell is allocated to another user and cannot be opened.

BUGS

It draws grid representations somewhat slowly; range images are produced VERY slowly.

HISTORY

01-Feb-83 Hannah at SRI-IU
Changed name from dtmplot to showdtm; installed latter on Testbed. Changed the argument parsing mechanism to permit a hands-off demo.

30-Nov-82 Hannah at SRI-IU
Modified code written at Lockheed to run on Grinnell.

01-Feb-82 Hannah at Lockheed Missiles and Space Company
Created.

NAME

stereo – the Moravec multiple image correlator

SYNOPSIS

stereo

DESCRIPTION

Stereo implements the Moravec correlation and rangefinding package. Two to nine images are accepted, and "interesting" points from one image are correlated with each other image. Distances are calculated for each point.

The user is prompted for the number of images to be processed.

The user is prompted for a filename base, from which individual
file names are formed by adding #.img, where # is a sequence number. The user can accept or change the filenames.

The user is prompted for the size of the window to be used by
the correlator.

FILES

/iu/tb/src/stereo/*

DIAGNOSTICS

Prints error message and quits with a return value of 1 for files not found.

BUGS

Check the source code header for suggestions.

HISTORY

01-Feb-83 Laws at SRI-IU
Stereo now accepts relative image names instead of requiring a full path specification. The search path is taken from the PICPATH environment variable, and defaults to ":/iu/tb/pic:/aux/tbpic".

01-Oct-81 Chuck Thorpe (cet) at Carnegie-Mellon University
Created.

NAME

view – formatted printout of a data file

SYNOPSIS

view [filename] [**-verbose**] [**-type** pixeltype] [**-header** hdrbytes] [**-dimensions** datacols datarows] [**-print** colwidth [ndecimals]] [**-n** wdwcols wdwrows] [**-from** mincol minrow]

DESCRIPTION

View produces formatted data printouts for files having a two-dimensional (or one-dimensional) structure. You specify the file name, the pixel type (byte, short, int, long, float, or double), the number of header bytes to skip, and the two-dimensional structure of the following data. (View -v will help you determine the rectangular dimensions.) You then specify the output column format, the size of window you want displayed, and the initial (or upper left) coordinate.

Any of these parameters may be specified on the command line or interactively. To force interaction, include the -v flag; otherwise view will try not to annoy you with dumb questions. If you omit a -t type specification, view will read the file as byte data.

If you use a -f flag on the command line, and you do not also specify -v, the program will print a single data window and then exit. Otherwise it will put you into an interactive loop. It will print data displays as long as you keep specifying initial points. To quit, either type ^C or \quit. The former will exit immediately, the latter will give you a chance to change your formatting options and try again.

Arguments are parsed using the arglib package. Each flag introduces a new "command line" to be parsed. Special arglib flag values such as "??" or "?>" may be used with each of these clauses.

FILES

/iu/tb/src/view/view.c

SEE ALSO

od(1), see(1), matrixlib(3)

DIAGNOSTICS

View warns you if the header length and dimensions you specify do not account for the entire data file. Other error messages may come from the invoked subroutines, and will generally be in printerr format.

BUGS

The name "view" has also been used at Berkeley for a variant of the vi editor.

Printout is inherently top-to-bottom, whereas Testbed image data is usually scanned bottom-to-top. Beware the inversion. Also note that view knows nothing of image blocking, padding, etc. It just reads bytes from the file.

You cannot specify a file format that extends beyond the physical file length. This makes view difficult to use for viewing incomplete files.

Output formatting is independent of the data values read. This can produce spectacular garbage if the pixel type specification is incorrect. It may also be wrong if the data contains special codes such as NOTINT or NOTFLOAT. Another problem is that the space required for the row numbers in the left margin is not adaptive or controllable.

Someday we may be able to guess the data type by examining the file. We could also try to guess the header length. If the file is an image file, we could open it for image I/O, which includes byte unpacking and deblocking. (Methods of overriding these defaults would still be needed.)

As clever as this control structure is, it is probably inferior to a command driver that lets the user set any parameter whenever he wants. Another promising interface is an EMACS-style editor; it should allow you to view and alter the data. Simultaneous display of image data on a video monitor would also be nice.

HISTORY

10-Mar-83 Laws at SRI-IU
Created.

NAME

whist – write a history message

SYNOPSIS

whist [-] [**-t** typename] files...

DESCRIPTION

Whist adds history messages to text files. It is useful for maintaining records of changes to source files and manual entries.

Normally, you enter a message for each file separately, using your favorite editor. If you wish, you may provide a single history message on the standard input; then, use the "-" flag to tell whist to use the standard input as the history message. This flag also inhibits all interaction with the user by whist.

With the -t flag, you may specify exactly what kind of file is being updated. This is most useful for programs that execute whist, so that whist does not have to guess the file type or ask the user.

Whist has some features that are unique to CMU: it understands how to update G source code, and it parses the "gecos" field of the password file in a manner useful only at CMU. The code that performs these CMU-related code is conditionally compiled with appropriate alternatives; if you do not define CMU, the CMU-specific portions will all drop away.

FILES

/iu/tb/src/whist/whist.c
/tmp/docXXXXXX.X

SEE ALSO

doc(1)

BUGS

Check the source code header for suggestions.

HISTORY

14-Dec-81 Laws at SRI-IU
Imported from CMU and converted to Testbed conventions.

19-May-81 John Zsarnay (jaz) at Carnegie-Mellon University
Made a number of changes. First of all, the header name is the logged-in name from utmp, rather than the effective user ID. Next, the action1 action prompt has been changed so that it is similar to the post program. Also, a "type" command has been added to type the current log entry. The update strategy now recreates the old file and copies the new file on top of it. The intermediate is created in the same directory as the source file to aid in crash recovery. A bug in which interrupts weren't turned off when updating was fixed by ignoring them during the critical period.

20-Feb-80 Steven Shafer (sas) at Carnegie-Mellon University
Added support for PASCAL files: FPASCAL.

29-Jan-80 Steven Shafer (sas) at Carnegie-Mellon University
LISP files are no longer unique to CMU (since lizst has been distributed by U.C. Berkeley); YACC files are now supported; error messages have been improved.

19-Dec-79 Steven Shafer (sas) at Carnegie-Mellon University
Added -t flag so that programs can dictate to whist exactly what kind of

file is being updated. In particular, the "doc" program sometimes produces manual entries with unrecognizable sufixes (such as "1m" or "3s").

13-Dec-79 Steven Shafer (sas) at Carnegie-Mellon University
Added "quoteq" parameter to copymsg() for shell command files; the shell cannot "ignore-rest-of-line" ever, so you need to put all arbitrary text (i.e., embedded punctuation marks) inside double quotes.

05-Dec-79 Steven Shafer (sas) at Carnegie-Mellon University
Removed .PP from trailer for manual entry history messages. It is not necessary, and doc(1) doesn't understand it.

PART 3 — Subroutines and Utility Libraries

[illegible]

NAME

arglib – command-line and interactive argument query package

SYNOPSIS

#include "arglib.h"

```
boolean boolarg(cmdarg,dftval,[askmsg]);
   boolean dftval;
   string askmsg;

PNT crsarg(cmdarg,limitwdw,dftval,[askmsg]);
   WDW limitwdw;
   PNT dftval;
   string askmsg;

double doublearg(cmdarg,minval,maxval,dftval,[askmsg]);
   double minval,maxval,dftval;
   string askmsg;

int flagarg(flag,[argc,argv]);
   string flag;
   list argv;

float floatarg(cmdarg,minval,maxval,dftval,[askmsg]);
   float minval,maxval,dftval;
   string askmsg;

IMGNME imgnmearg(cmdarg,flagvect,dftstr,[askmsg])
   boolean flagvct[5];
   string dftstr,askmsg;

int intarg(cmdarg,minval,maxval,dftval,[askmsg]);
   string askmsg;

int openarg(argc,argv,[minargs,maxargs],usagestr);
   list argv;
   int minargs,maxargs;
   string usagestr;

int openflag(flag,argc,argv,[minargs,maxargs],usagestr);
   string flag;
   list argv;
   int minargs,maxargs;
   string usagestr;

int optionarg(cmdarg,optionlist,dftstr,[askmsg]);
   list optionlist;
   string dftstr,askmsg;

PNT pntarg(cmdarg,limitwdw,dftpnt,[askmsg]);
   WDW limitwdw;
   PNT dftpnt;
   string askmsg;
```

string stringarg(cmdarg,dftval,[askmsg]);
  string dftval,askmsg;

WDW wdwarg(cmdarg,limitwdw,dftwdw,[askmsg]);
  WDW limitwdw, dftwdw;
  string askmsg;

DESCRIPTION

ARGLIB is a library of routines for decoding command-line arguments or for obtaining values interactively from the user.

Openarg() or openflag() may be used to initialize the argument decoding process and must be present before the other routines may access the command-line arguments. (Initialization is not required if the other arglib routines are used only for interactive argument queries.) Openarg() makes the entire argument list available, whereas openflag() opens only the sublist associated with a specified flag.

Openarg() makes copies of the specified argument count (argc) and of pointers to the argument list (argv) and the usage string (usagestr). It parses the argument list to determine the number of arguments following argv[0] and before the first flag, and compares this number to minargs and maxargs (which default to 0 and INF). If the number (adjusted for any "?> extension) is outside this range, openarg() prints a usage message and terminates program execution. (The same action is taken if the first command-line argument is "??", since this is assumed to be a usage query.)

Openarg() returns the number of arguments, including argv[0], up to the first flag. This number may be used as an "argc" to again call openarg and limit arglib parsing to the initial list, if desired. (Another way to limit parsing to this substring is to call openflag() with the name of the main program used as a flag.) Otherwise the entire list, including all flags lists, will be available to the arglib parsing routines.

Openflag() is similar, except that it first skips to the specified flag argument. (The "flag" string is matched against the leftmost characters of each argument. It will usually include a "-" as the first character.) Further arglib parsing is restricted to the flagged sublist, and the length of this list (including the initial flag) is returned. If the flag is not present, openflag() returns 0 and restricts further parsing to a null list.

Flagarg() provides another way to test for the presence of a flag. It returns the argument number of the specified flag, or -1 if the flag is not present. Both openflag() and flagarg() also set the printerr global status code to WARNING if the flag is missing.

The remaining routines are all used to obtain argument values from the program command line or from the user. The basic function call is

```
 fooarg(cmdarg,legalvals,dftval,askmsg),
```

where the legalvals specification depends on the datatype.

If cmdarg == NOTINT, the user will be asked for a value without regard to any command-line values. Otherwise the action depends on the command-line argument corresponding to abs(cmdarg).

If the command-line argument is a "?", or if it is omitted but the last argument given was "?>", the user will be queried for a value. If the command-line argument had any other valid value, that value will be returned. If it had an invalid value, the user will be queried for a replacement.

If the argument was omitted and the last specified argument was not "?>", the action depends on the sign of cmdarg. If cmdarg is negative, the user will be asked for a value; otherwise the default will be returned silently. (An error message will be printed if no default was supplied. No message will be printed for a default outside the legalvals range unless the printerr OK flag has been set.)

Default values are supplied in the subroutine calls. Usually the default will be a legal response. The programmer may specify that there is no acceptable default by using NOTINT for integer and boolean defaults, NOTFLOAT or NOTDOUBLE for float or double defaults, or NULL for strings or blklib structures. (These macros are defined in the testbed.h include file.) The programmer may also supply an illegal default – out of range for a numeric, or "" for a string – which will be returned only if the user types a carriage return to accept the default; this can be useful as a "none" response.

The prompt string and default are printed whenever a value must be requested. (The notation "<default>" is used for a valid default, "<>" for a special or invalid default, and ">" for no default.) If the user just types a carriage return, the default will be returned. (This response will be rejected if there is no default.) The user may also type "?" to request information about legal responses.

If you do not specify the prompt string, the arglib routine will use a default prompt message suitable for the datatype. To suppress all messages, pass a NULL value for askmsg. For even greater control, askmsg may be a LST of strings; an initial string to be printed the first time the user is queried; a set of prompt strings for the various structure fields; and a set of help messages in case the user responds to a prompt with a "?". Elements of the LST that are omitted will revert to the standard defaults.

Each routine returns the appropriate datatype (as declared in arglib.h) and a printerr-style global status code. The returned value will either be a command-line value; the default; a value typed by the user; or (in the case of a "\quit" response from the user) NULL, NOTINT, or similar missing value code.

Arglib subroutines:

boolarg(cmdarg,dftval,[askmsg]) -> boolean
: Obtain a yes/no, true/false, or 1/0 argument value.

crsarg(cmdarg,limitwdw,dftval,[askmsg]) -> PNT
: Use the keyboard cursor to get a point specification. If either coordinate is given on the command line, even if invalid or a "?" query flag, both

values will be obtained via pntarg() instead of with the cursor. Be sure that openarg() has been called if cmdarg is not NOTINT, and be sure that the display has been initialized (e.g., with g_init(0)) if cmdarg is negative.

doublearg(cmdarg,minval,maxval,dftval,[askmsg]) -> double
Obtain a double-precision argument value.

flagarg(flag,[argc,argv]) -> int
Returns the argument number of the specified flag, or -1 if the flag is not present; also sets the printerr global status code to WARNING if the flag is missing.

floatarg(cmdarg,minval,maxval,dftval,[askmsg]) -> float
Obtain a floating-point argument value.

imgnmearg(cmdarg,flagvect,dftstr,[askmsg] -> IMGNME
Obtain an IMGNME image name argument.

intarg(cmdarg,minval,maxval,dftval,[askmsg]) -> int
Obtain an integer argument value.

openarg(argc,argv,[minargs,maxargs],usagestr) -> int
Initialize the argument decoding process so that other arglib routines may access the command-line arguments. It parses the argument list to determine the number of arguments following argv[0] and before the first flag, and compares this number to minargs and maxargs (which default to 0 and INF). If the number (adjusted for any "?> extension) is outside this range, openarg() prints a usage message and terminates program execution. (The same action is taken if the first command-line argument is "??", since this is assumed to be a usage query.) Returns the number of arguments, including argv[0], up to the first flag.

openflag(flag,argc,argv,[minargs,maxargs],usagestr) -> int
Initializes the sublist associated with a specified flag and returns the length of this list (including the initial flag). If the flag is not present, openflag() returns 0 and restricts further parsing to a null list.

optionarg(cmdarg,optionlist,dftstr,[askmsg]) -> int
Obtain the optionlist index of a string argument.

pntarg(cmdarg,limitwdw,dftpnt,[askmsg]) -> PNT
Obtain a PNT coordinate pair argument.

stringarg(cmdarg,dftval,[askmsg]) -> string
Obtain a string argument value.

wdwarg(cmdarg,limitwdw,dftwdw,[askmsg]) -> WDW
Obtain a WDW window argument.

EXAMPLE

```
#include "arglib.h"

main(argc,argv)
 int argc;
 list argv;
{

 float val1,val2,val3;

 /* Initialize the argument package. */
 openarg(argc,argv,"prntvct val1 val2 [scale]");

 /* Read two components and query missing values. */
 val1 = floatarg(-1,-1.0,1.0,0.0,"First component:");
 val2 = floatarg(-2,-1.0,1.0,0.0,"Second component:");

 /* Unit scaling if not given on the command line. */
 val3 = floatarg(3,-FLOATINF,FLOATINF,1.0,"Scale factor:");

 /* Print the vector. */
 if (boolarg(NOTINT,TRUE,"Print the vector?")) then
    printf("  (%g,%g)0,val1*val3,val2*val3);
}
```

For an example of flag argument parsing see the "view" program source code.

FILES

/iu/tb/include/arglib.h

/iu/tb/lib/sublib.a
/iu/tb/lib/sublib/arglib/*
/iu/tb/lib/sublib/pntlib/pntarg.c
/iu/tb/lib/sublib/wdwlib/wdwarg.c

/iu/tb/lib/imagelib.a
/iu/tb/lib/imagelib/gmrlib/crslib/crsarg.c
/iu/tb/lib/imagelib/imgnmelib/imgnmearg.c

External variables:
```
 int savedargc;
 list savedargv;
 string savedusage;
 boolean savedquery;
```

SEE ALSO

asklib(3), imgnmelib(3), parselib(3), pntlib(3), wdwlib(3)

DIAGNOSTICS

This package uses the printerr system to communicate the type of user interaction that has taken place. The global error code will be OK if there is no interaction with the user, WARNING if there has been interaction, and QUIT if the user has forced a return by responding with "\quit". The only ERROR condition is

EOF, and the routines call exit(QUIT) rather than returning. (Typing ^D to a query also simulates an EOF condition and terminates program execution.) This action is designed to terminate improper batch scripts so that they don't eat CPU time and spit garbage.

If you use "boolean", "string", or "list" declarations, as above, you must also include "arglib.h" or "testbed.h". Otherwise the compiler will complain.

BUGS

Beware of the "\quit" return. The user is advised not to use this escape on any program that doesn't advertise its availability. Programmers are advised to write code that checks whether a NULL, NOTINT, or similar code has been returned. A future version of the arglib code may use a global variable to allow or disallow quit returns.

A planned extension is to allow a STR dynamic string to take the place of cmdarg. Arglib routines will then be able to take over the asklib functions.

Optionarg() has inherited a few inconveniences from bestmatch(); see the listlib **man** page for a discussion.

See the source directory for additional comments.

HISTORY

08-Dec-83 Laws at SRI-IU
Added crsarg().

22-Mar-83 Laws at SRI-IU
The imgnmelib.h, pntlib.h, and wdwlib.h include files are now brought in by arglib.h.

04-Mar-83 Laws at SRI-IU
Made the openarg() argument counts optional. Changed openarg() to count arguments up to the first flag. Added openflag() and flagarg().

17-Jan-83 Laws at SRI-IU
Added IMGNME, PNT, and WDW datatypes. Deleted imgarg(), which was not written in blklib style.

26-Jul-82 Laws at SRI-IU
Wrote initial version.

NAME

asklib – ask user to type a value

SYNOPSIS

```
#include "asklib.h"

int askbool (prompt,defalt);
   char *prompt;
   int defalt;

int askchar (prompt,legals,defalt);
   char *prompt,*legals,defalt;

double askdouble (prompt,min,max,defalt);
   char *prompt;
   double min,max,defalt;

float askfloat (prompt,min,max,defalt);
   char *prompt;
   float min,max,defalt;

unsigned int askoctal (prompt,min,max,defalt);
   char *prompt;
   unsigned int min,max,defalt;

unsigned int askhex (prompt,min,max,defalt);
   char *prompt;
   unsigned int min,max,defalt;

int askint (prompt,min,max,defalt);
   char *prompt;
   int min,max,defalt;

long asklong (prompt,min,max,defalt);
   char *prompt;
   long min,max,defalt;

short askshort (prompt,min,max,defalt);
   char *prompt;
   short min,max,defalt;

char *askstring(prompt,defalt,buffer);
   char *prompt,*defalt,*buffer;

int asklist(prompt,optlst,defalt);
   char *prompt,**optlst,*defalt;

int askoption(prompt,optlst,defalt);
   char *prompt,**optlst,*defalt;
```

DESCRIPTION

These routines are used primarily in the CI command interpreter and in main programs that invoke that interpreter. For other purposes, please see arglib(3). These asklib routines may eventually be eliminated when experience has been gained in converting CI routines to arglib form.

Askbool is used to ask the user to type "yes" or "no". *Prompt* is a string containing a message that will be printed; the user types "yes", "y", etc., or "no", "n", etc. in response to the question. *Askbool* returns 1 if "yes" (etc.) is typed, 0 for "no". If the user types just a carriage return, then *defalt* is returned.

Askchar will print its prompt and ask the user for a single character. If that character is in the string of legal response characters (*legals*), then its index is returned. If a null string is typed (just a carriage return), then the default character (*defalt*) will be assumed and its index returned. If the character typed is not in the string of legal responses, then an error message is printed and the prompt-and-response cycle is repeated.

Askdouble and *askfloat* ask the user to type a floating-point number. They begin by printing the string *prompt* as a message to the user. The user then types in a floating-point number that is parsed. If the number is valid and is within the range *min* to *max*, inclusive, then the value is returned as the value of *askdouble* or *askfloat*. If the value is invalid or out of range, then an error message is printed and the cycle is repeated. If the user types just a carriage return, the value *defalt* is returned.

Askdouble and *askfloat* are identical, except for the type of the parameters and the results.

Askoctal and *askhex* ask the user to type in an unsigned octal or hexadecimal integer.

They begin by printing the string *prompt* as a message to the user. The user then types in a number of the appropriate form. If the number is valid and is within the range *min* to *max*, then it is returned as the value of *askoctal* or *askhex*. If it is invalid or is out of range, then an error message is printed and the prompt-and-response cycle is repeated. If the user types just a carriage return, then the value *defalt* is assumed.

Askoctal and *askhex* are identical, except in the manner in which the user's input is converted into an unsigned integer.

Askoctal converts the input by assuming it represents a string of octal digits. Leading blanks and tabs are ignored; conversion stops at the first character that is not a legal octal digit. No signs (+ or -) are allowed.

Askhex converts the input by assuming it represents a string of hexadecimal digits. There may be leading blanks and tabs; in addition, the digit string may be preceded by "0x" or "0X", which will be ignored. The valid characters include "0" through "9", "a" through "f", and "A" through "F". Conversion stops at the first character that is not a legal hexadecimal digit. No signs (+ or -) are allowed.

Askint, *asklong*, and *askshort* ask the user to type in an integer. They begin by printing the string *prompt* as a message to the user. The user then types in a number. If the number is valid and is within the range *min* to *max*, then it is returned as the value of *askint (asklong, askshort).* If it is invalid or is out of range, then an error message is printed and the prompt-and-response cycle is repeated. If the user types just a carriage return, then the value *defalt* is assumed.

Askint, *asklong*, and *askshort* are identical, except for the type of the parameters and the results.

Askstring will print *prompt* as a message to the user, and ask for the user to type an input string. The string is then copied into *buffer.* If the user types just a carriage return, then the string *defalt* is copied into *buffer. Buffer* and *defalt* may be the same string; in this case, the default value will be the original contents of *buffer.*

Asklist will print *prompt* as a message to the user, and ask for an input string. The string typed will be sought within the string optlst *optlst*, and, if found, the index of this string will be returned. If the user types just a carriage return, then the string *defalt* will be assumed. If nothing in the optlst matches the input string, or if more than one string matches, then an error message is printed and the prompt-and-response cycle is repeated. The string optlst may be declared this way:

```
char *optlst[] = {
        "first string",
        "second string",
        ...
        "n-th string",
        0};
```

Askoption is just like *asklist*, but performs a heuristic test for the quality of each string match and returns the index of the best match. If the match is not exact, the user is asked to approve or disapprove the choice of the matching string; if he disapproves, several other closely matching choices are listed.

FILES

/iu/tb/include/asklib.h
/iu/tb/lib/sublib/asklib/*
/iu/tb/lib/sublib.a

SEE ALSO

arglib(3), parselib(3), listlib(3), stringlib(3)

DIAGNOSTICS

The user is informed if his response is not valid and the prompt-and-response cycle is repeated.

BUGS

There is no provision for an "escape" answer (e.g., "Quit and return to command driver level.").

No provision is made for detecting or correcting overflow.

HISTORY

26-Nov-82 Laws at SRI-IU
Combined the individual **man** files into this one.

21-Dec-81 Laws at SRI-IU
: Changed names from getbool to askbool, getchr to askchar, getdouble to askdouble, getfloat to askfloat, getoct to askoctal, gethex to askhex, getint to askint, getlong to asklong, getshort to askshort, getstr to askstring, getstab to asklist, and getsearch to askoption.

23-Jan-80 Steven Shafer (sas) at Carnegie-Mellon University
: Getlst added.

05-Dec-79 Steven Shafer (sas) at Carnegie-Mellon University
: Created.

NAME

blklib – object-oriented structure manipulation

SYNOPSIS

#include "blklib.h"

Block structure (pertinent fields only):

```
typedef BLK * struct {
     BLKTYPE idcode;
     int linkcount;
     pointer proplist;
     ...
};
```

BLK closeblk(blk)
BLK blk;

BLK copyblk(inblk)
BLK inblk;

BLK linkblk(blk)
BLK blk;

printblk(blk)
BLK blk;

DESCRIPTION

The object-oriented style of programming used in the SRI Image Understanding Testbed requires that there be an identifier field, link count, and property list as the first elements of every data block. This file describes routines for manipulating data blocks by using these fields. There are currently no routines for manipulating property lists.

Since the BLK datatype itself is not very useful, the user should cast the values returned by these routines to the appropriate object type. The advantage of using these general blklib routines over similar type-specific routines (e.g., closewdw) are solely in the user interface: a command driver may "close" any blklib object without itself testing to determine which routine is appropriate.

Block identifier codes.
The following object types have been implemented:

```
typedef enum {
   NOID = 0,            Illegal ID code.
   DSPID = 99901,       Display descriptor.
   FRMID,               Display frame descriptor.
   HDRID,               Picture header descriptor.
   IMGID,               Image descriptor.
   IMGNMEID,            Image name descriptor.
   LSTID,               List descriptor.
   PICID,               Picture file descriptor.
   PNTID,               Point descriptor.
   SEGID,               Line segment descriptor.
   WDWID                Window descriptor structure.
} BLKTYPE;
```

To add other datatypes, define the appropriate structures in an include file (as in /iu/tb/include/wdwlib.h), add the ID code to the blklib.h file, and (optionally) add the appropriate switching code to the routines listed here. You must also write the datatype manipulation routines, of course; copying an existing blklib directory and editing the datatype names is usually a good way to begin.

Blklib routines:

These are "big switch" routines that call the appropriate special-purpose routine for the passed block. They must be linked in ahead of any of the invoked primitive object handlers (e.g., closepic()). To make this more convenient, two blklibs have been defined: one in /iu/tb/lib/blklib for the simple datatypes defined there, and one in /iu/tb/lib/visionlib for higher level datatypes such as PICs, IMGs, IMGNMEs, DSPs, and FRMs. Normal linking procedures will give you the needed versions of the following utilities:

closeblk(blk) -> BLK
    Invoke the closing routine for the given block type.

copyblk(inblk) -> BLK
    Invoke the copying routine for the given block type.

linkblk(blk) -> BLK
    Increment the link count of the block and return its pointer.

printblk(blk)
    Invoke the printing routine for the given block type.

FILES

/iu/tb/include/blklib.h
/iu/tb/lib/sublib/blklib/*
/iu/tb/lib/sublib.a

Optional Top-Level Blklib:
/iu/tb/lib/visionlib/blklib/*
/iu/tb/lib/visionlib.a

SEE ALSO

dsplib(3), frmlib(3), hdrlib(3), imglib(3), imgnmelib(3), lstlib(3), piclib(3), pntlib(3), seglib(3), wdwlib(3)

DIAGNOSTICS

The blklib routines use the printerr error reporting system. An error will typically cause a message to be printed on the error stream and the routine will return a null BLK.

BUGS

The blklib library itself distributes the code that must be maintained whenever an object type is altered. It is suggested that users avoid these general routines unless they have a very good reason for using them. The blklib style of object-oriented programming is definitely useful, however,and should be embraced wholeheartedly.

Linkblk is currently not a "big switch" routine; it just manipulates the linkcount field without regard to the block type. There may be a problem here since this is not what the linkimg() routine in imglib currently does.

HISTORY

01-Feb-83 Laws at SRI-IU
Documented the new proplist field and the dual blklib implementation. Added the IMGNME, LST, PNT, and SEG datatypes.

17-Nov-82 Laws at SRI-IU
Created this file.

NAME

ci − command interpreter

SYNOPSIS

```
#include <stdio.h>
#include "ci.h"

ci(prompt,file,depth,list,helppath,cmdfpath);
   char *prompt,*helppath,*cmdfpath;
   FILE *file;
   int depth;
   CIENTRY *list;
```

DESCRIPTION

CI allows the user to type instructions that execute commands and display and assign values to variables. The commands that are executed correspond to procedures supplied by the user's program; the variables correspond to variables in the program. The most basic facilities of *CI* are easily learned and used, and it provides other, more sophisticated, features that make it suitable for complex command-driven programs.

CI has a basic loop that consists of printing the string *prompt*, reading a line from its current input file, and executing the specified instructions. The formats and meanings of these instructions are as follows:

command [arg ...]
: Execute the procedure that corresponds to the named command. The list of arguments will be passed to this procedure. See details below.

variable [param ...]
: Display the value of the named variable. Some variables may require parameters; see details below.

variable = [param ...] value
: Assign the given value to the named variable. The equals sign (=) may appear anywhere in the line.

command*
: List the names of commands that begin with "command."

variable*=
: List the names and values of variables that begin with "variable."

<commandfile
: Recursively call *CI*, with the new procedure reading its commands from "commandfile".

?topic
help topic
: Print help message related to topic; or, if topic matches more than one help message name, list the names of matching topics. If no topic is specified, a standard help message describes how to use *CI is displayed*. If the topic is "*," a list of help topics is produced.

!
!shellcommand
: Fork a shell; if a command is specified, execute it and have the shell exit.

:comment

Ignore this instruction.

^D (control-D)
Exit from *CI.*

Several instructions may appear on the same line, separated by semicolons. In addition, *CI* allows abbreviation of command names, variable names, and help topics to any unique prefix.

PARAMETERS TO CI

Prompt is the string that is printed to tell the user to type a command line to *CI*. During execution of command files, the prompt will be echoed along with each line read from the file; both will be indented to indicate that this command was read from a command file.

File is the file to be used for input. Normally, the standard input *(stdin)* is used; if you specify 0, *CI* will assume you are using the standard input.

Depth is used for indenting the prompt; this should usually be 0 when you execute *CI*, but when it calls itself recursively for command files, the depth will be nonzero.

List is an array of the entry descriptors for commands and variables; you declare this to be of type *CIENTRY* (see below), and use the macros described below to describe the entries.

Helppath is a list of directories containing help files, separated by colons. If you specify a single directory, that's fine; *CI* will assume all help files lie in that directory. When the user asks for help on some topic, *CI* looks for a file in one of the help directories that has a matching name. The contents of the file will simply be typed out to the user. If you specify 0 for ***helppath***, *CI* will assume that no help files exist.

Cmdfpath is a list of directories containing command files, separated by colons. This is useful for libraries of prepared command files. The current directory should be included in the list; this is best done by indicating a null directory name (i.e., a colon as the first character of the path, or two consecutive colons within the path). If you specify 0 (which you will probably do most of the time), *CI* will assume that all command file names are evaluated with respect to the current directory only. Absolute pathnames, of course, are always valid.

THE ENTRY LIST

The parameter *list* is a list of objects of type *CIENTRY*. These objects, defined by a set of macros, consist of a name which is a character string, a pointer to a value, and a type. You declare the list in this manner:

```
CIENTRY list[] = {
        CIINT(...),
        CIDOUBLE(...),
        CICMD(...),
        ...
        CIEND
};
```

The macros that define entries are described below.

COMMANDS

A *command* is a procedure provided by your program that can be executed by a user by typing its name and, optionally, a list of arguments. You specify a command by providing the procedure, which must take a character string as an argument, and by placing an entry into the CIENTRY list:

```
mycommand (arglist)
char *arglist;
{
        char *p;        /* recommended for parsing */
        int arg1,arg2;
        p = arglist;
        arg1 = intarg (&p,0,...);       /* see intarg(3) */
        arg2 = intarg (&p,0,...);
        ...
}
...
CIENTRY list[] =
{
        ...
        CICMD ("munch",mycommand),
        ...
        CIEND
};
```

The user can then type "munch 3 4," and myproc will be executed with arglist equal to "3 4". The parsing sequence shown above (using *intarg*(3)), will assign 3 to arg1 and 4 to arg2. If the user were to type "munch" with no arguments, he would be prompted for arg1 and arg2 as described in *intarg*(3).

SIMPLE VARIABLES

CI knows how to manipulate several kinds of simple variables. To use these, you declare a variable of the appropriate type and place an entry into the CIENTRY list. The types of variables known to *CI* correspond to the macros that you put into the list:

CIINT ("name", variable)
: This specifies a variable of type *int*. "Name" is the name of the variable as it will appear to the user who is executing the program.

CILONG ("name", variable)
CISHORT ("name", variable)
: These specify variables of type *long* (actually, *long int)* and *short* (actually, *short int)*, respectively.

CIOCT ("name", variable)
CIHEX ("name", variable)
: These specify *unsigned int* variables, whose values will be shown and interpreted as octal and hexadecimal integers, respectively. Thus, the value of an octal variable might be 07773; the value of a hexadecimal variable might be 0xabc.

CIDOUBLE ("name", variable)
CIFLOAT ("name", variable)
: These indicate floating-point variables of types *double* and *float*, respectively.

CIBOOL ("name", variable)
: This indicates a variable of type *int*, whose value will be either "yes" (i.e., nonzero), or "no" (i.e., zero).

CISTRING ("name", variable)
: This indicates a variable of type "char *" or "char []", which will be treated as a character string. For *CI* to work properly, this string should not contain garbage when you call *CI*.

Here is an example of two variables and how they might be used:

```
int i;
char s[100];
...
CIENTRY list[] =
{
        CIINT ("number",i),
        CISTRING ("string",s),
        ...
        CIEND
};
```

Here is an excerpt of a dialogue with the program containing the above statements (lines typed by the user are indicated by italics):

```
n=3
number          3
s=Hello, mom!
string      Hello, mom!
nu
number          3
num=4
number          4
*=
number          4
string      Hello, mom!
```

CLUSTER VARIABLES

In addition to the simple variables described above, *CI* can manipulate "clustered variables," which consist of a variable and some descriptive information about it. The descriptive information for a variable of type X (int, float, etc.) is exactly the information in the parameter list of the routine called "getX" (*askint*(3), *askfloat*(3), etc.). It typically includes some description of the legal values for the variable and a prompt string printed to remind the user what this variable means.

To use a clustered variable involves two steps: you must declare the variable itself, together with its description; and you must insert the proper declaration into the CIENTRY list.

To declare a clustered "int" variable, use this macro:

```
        CINT (sname, vname, min, max, "prompt");
```

This macro appears just like any other declaration, but must be outside of any procedures (i.e., global). It will create an int variable called *vname*, which you may refer to in other parts of your program; it also declares a structure called *sname* that contains the description of *vname*. The description consists of three values: *min*, the minimum allowable value for *vname; max*, the maximum allowed value; and *prompt*, the prompt string for assigning a value to *vname*.

The corresponding entry of the CIENTRY list would be:
    CICINT ("name," sname)
where *sname* is the same as *sname* in the CINT macro.

A clustered variable differs from a simple variable in two ways. When a user tries to assign a value to a clustered variable, the new value is checked for legality. If it is legal, it is assigned; otherwise, a message is printed and the user can type another value. Also, the user may type "name=", omitting the value, and will be prompted for the value to be assigned.

Here are the clustered types known to *CI:*

CINT (sname, vname, min, max, "prompt");
CICINT ("name", sname)
    Declares a clustered int variable. The legal range of values is [min..max]. The variable will be called *vname*. As indicated above, CINT is a declaration, and CICINT is the corresponding entry in the CIENTRY list.

CLONG (sname, vname, min, max, "prompt");
CICLONG ("name", sname)
CSHORT (sname, vname, min, max, "prompt");
CICSHORT ("name", sname)
    These define long and short clustered variables, respectively. CLONG and CSHORT are the declaractions; CICLONG and CICSHORT are the entries for the CIENTRY list.

COCT (sname, vname, min, max, "prompt");
CICOCT ("name", sname)
CHEX (sname, vname, min, max, "prompt");
CICHEX ("name", sname)
    These define unsigned int clustered variables whose values are interpreted as octal or hexadecimal numbers, respectively. COCT and CHEX are declarations; CICOCT and CICHEX are CIENTRYs.

CDOUBLE (sname, vname, min, max, "prompt");
CICDOUBLE ("name", sname)
CFLOAT (sname, vname, min, max, "prompt");
CICFLOAT ("name", sname)
    These define floating-point variables (double and float, respectively). CDOUBLE and CFLOAT are declarations; CICDOUBLE and CICFLOAT are CIENTRYs.

CBOOL (sname, vname, "prompt");
CICBOOL ("name", sname)
    Defines an int variable whose value is interpreted as "yes" (nonzero) or "no" (zero).

CCHR (sname, vname, "legals", "prompt");
CICCHR ("name", sname)
    Defines an int variable whose value corresponds to a single character within the string *legals.* The value will be printed as the character

indexed by the current value of the variable (i.e., *legals*[*vname*]), and when assigning a value to it, the user types a character. The index of that character within *legals* will then be assigned to *vname*.

CSTRING (sname, vname, length, "prompt");
CICSTRING ("name", sname)

These define a variable that is a character array of length *length*. It will be treated as a character string.

CLIST (sname, vname, optlst, "prompt");
CICLIST ("name", sname)

These define a variable of type int that corresponds to one of the strings in the string list *optlst*. The list is declared as for *asklist*(3). The value is displayed as the string it indexes (e.g., *list*[*vname*]), and to assign a value, the user types a string that matches an entry of the table.

Here is an example using two clustered variables:

```
CINT (si, i, 1, 10, "What's your favorite number?");
CSTRING (sname, name, 100, "What's your name?");
...
CIENTRY list[] =
{
        CICINT ("favorite",si),
        CICSTRING ("name",sname),
        ...
        CEND
}
```

This might be part of a dialogue with the program containing the above declarations (lines typed by the user are indicated in italics):

```
fav=7
favorite      7
name=Humpty Dumpty
name          Humpty Dumpty
fav=32
32 out of range.  What's your favorite number?  (1 to 10)  [7] 4
favorite      4
f=
What's your favorite number?  (1 to 10)  [4]  8
favorite      8
name=
What's your name?  [Humpty Dumpty]  Minnie Mouse
name          Minnie Mouse
*=
favorite      8
name          Minnie Mouse
```

Most users, for most programs, will find clustered variables to be preferable to simple variables.

PROCEDURE VARIABLES

CI allows you to specify any type of variable you want – an ordered pair, a character font, a buffer of a color TV display, a strange plotter, a robot arm, a file, the color of the pajama tops worn by three hippopotami in the CS lounge, absolutely anything at all!

There is a catch, however. You have to write the procedure that manipulates the variable.

This type of variable is called a *procedure variable*. It consists of a procedure, which you must provide, and an entry on the CIENTRY list that looks like this:

```
        CIPROC ("name", procname)
```

where *procname* is the name of your procedure.

Your procedure will be called with two parameters:

```
        proc (mode,arglist)
        CIMODE mode;
        char *arglist;
```

The first parameter, *mode*, indicates what *CI* is trying to do; the second, *arglist*, is the list of parameters and values typed by the user.

The *mode* parameter may have one of three values:

CISET *CI* is trying to assign a value to the variable; i.e., the user typed "name=" or "name=value" or something like that.

CISHOW
CI is trying to display the value of the variable; i.e., the user typed "name".

CIPEEK
CI is trying to do a one-line printed display of the variable in the format "name<TAB><TAB>value". This is normally performed when the user types "*=", and you should do this following a CISET.

Typically, the procedure will use a *switch* statement to deal with the three cases. If the value can be displayed by printing it on one line, the CISET and CIPEEK cases may be the same. This is true, for example, for an ordered pair of integers; it is not true, say, for a variable that represents a color picture (to display this may involve writing it onto a color TV monitor).

Here is an example of a procedure variable that represents an ordered pair:

```
        int x,y;
        ...
        xy (mode,arg)
        CIMODE mode;
        char *arg;
        {
                char *p;                /* for parsing */
                switch (mode) {
                case CISET:
                        p = arg;
                        x = intarg (&p,0,"X coordinate?",-100,100,x);
                        y = intarg (&p,0,"Y coordinate?",-100,100,y);
                        /* now, fall through to display the value */
                case CISHOW:
                case CIPEEK:
```

```
                printf ("point\t\tx %d\ty %d\n",x,y);
        }
}
...
CIENTRY list[] =
{
        CIPROC ("point",xy),
        ...
        CIEND
}
```

Here is an example of dialogue with the program containing the above code (lines typed by the user are indicated by italics):

```
point=3 5
point           x 3     y 5
p=
X coordinate?  (-100 to 100)  [3]  72
Y coordinate?  (-100 to 100)  [5]  39
point           x 72    y 39
p= 287
287 out of range.  X coordinate?  (-100 to 100)  [72]  28
Y coordinate?  (-100 to 100)  [39]  29
point           x 28    y 29
p
point           x 28    y 29
```

Note that some kinds of variables may require parameters just to be displayed; you will receive a (possibly null) argument list every time your procedure variable is called, and you may parse arguments for all three activities specified by *mode*.

CLASS VARIABLES

On occasion, you may want to have several procedure variables that require the exact same code for their processing. For example, you may have sixteen different robot arms that you want the user to treat as variables; or you may have several windows on the color TV screen that you want to treat as variables. In such cases, it would be a shame to have to create several procedure variables, each with exactly the same code. To eliminate this duplication, *CI* provides a facility called the *class*.

A class is a collection of procedure variables that share the same code. Each variable, however, is distinguished by its own data. The entries on the CIENTRY list look like this:

```
        CICLASS ("name1",var1,classname),
        CICLASS ("name2",var2,classname),
```

and so on, one entry for each variable. *Var1* and *var2* are the names of the data areas for the variables; they might be declared like this:

```
        typedef struct { int field1; ... } DATAAREA;
        DATAAREA var1, var2;
```

Classname is the name of the procedure that is used by these variables for displaying and assigning a value.

The procedure will be called with four parameters. To continue the above example, the procedure might begin like this:

```
classname (mode,arglist,varptr,varname)
CIMODE mode;
char *arglist, *varptr, *varname;
{
        DATAAREA *p;
        . . .
        p = (DATAAREA *) varptr;
        . . . p->field1 . . .
```

In this example, note that the first two parameters are just the same as the first two parameters for a procedure variable. They have exactly the same meaning. The third parameter is a pointer to the data area for the variable being displayed or assigned to. This value must be of type "char *" for C's type-checking to work properly, so you will want to coerce it by a type-cast to be a pointer to the proper type. Note also that *var1* and *var2* are DATAAREAs, not (DATAAREA *)s. In general, whatever type of object you declare in the CICLASS macro, the parameter passed to the procedure will be a pointer to that type of object. The fourth (last) parameter passed to the procedure will be the name of the variable being processed, given in a character string.

Here is an example of two ordered pairs represented by two class variables:

```
typedef struct {int x,y;} ORDPAIR;
ORDPAIR startp,endp;
...
ordproc (mode,arg,cdata,name)
CIMODE mode;
char *arg, *cdata, *name;
{
        char *p;
        ORDPAIR *data;
        data = (ORDPAIR *) cdata;
        switch (mode) {
        case CISET:
                p = arg;
                data->x = intarg (&p,0,"X coordinate?",-100,100,data->x);
                data->y = intarg (&p,0,"Y coordinate?",-100,100,data->y);
        case CISHOW:
        case CIPEEK:
                printf ("%s\t\tx %d\ty %d\n",name,data->x,data->y);
        }
}
...
CIENTRY list[] =
{
        CICLASS ("start",startp,ordproc),
        CICLASS ("end",endp,ordproc),
        ...
        CIEND
}
```

Here is an example of a dialogue with the program containing the above code (lines typed by the user are indicated by italics):

```
start = 3 5
start           x 3     y 5
end = 6 10
end             x 6     y 10
start =
X coordinate?  (-100 to 100)  [3]  72
Y coordinate?  (-100 to 100)  [5]  39
start           x 72    y 39
```

INTERRUPT HANDLING

If you use *del*(3) to trap interrupts, you will receive a bonus from *CI*. If you hit DEL during the execution of a command, that command may trap it (by *DELRETURN*, etc.); if the command ignores it, *CI* will deal with it when the command is finished executing.

If the interrupt occurred while *CI* was reading from the standard input, it will just print "Interrupt ignored".

If, however, the interrupt occurred during a command file, *CI* will print:

```
INTERRUPT: Abort or breakpoint? [abort]
```

and wait for you to type something. If you type "a," or "abort," or just a carriage return, *CI* will abort the command file (i.e., pretend it just encountered the end of the file). If you type "b," or "breakpoint," or something like that, then *CI* will recursively call itself, with the new *CI* taking input from the standard input (e.g., terminal). The prompt will be "Breakpoint for *prompt*", where *prompt* is the prompt for the interrupted command file. When you exit from the new *CI*, the command file will be resumed as if nothing had happened.

EXTERNAL VARIABLES

CI uses three external variables (declared in the file <*ci.h*>) that you may also use in your program.

FILE *ciinput;

This variable is the current input file for *CI*. You can read lines from this file within your commands and variables, if you want to read from the same place that *CI* is reading from.

int ciexit;

If you put a nonzero value into this variable, *CI* will exit (i.e., return) when the current command (or procedure variable) is finished. This allows you to write a command that causes *CI* to exit.

int ciquiet;

This word contains several bits that govern the output produced by *CI*.

The bits for *CIquiet* are also declared in the macro file. If a bit is 0, the output will be produced; if it is 1, the output is suppressed.

CISHEXIT

Print a message when *CI* resumes after a shell command (i.e., "!" commands).

CISETPEEK

Automatically display the new value of a variable when that variable has a new value assigned. This effectively performs a CIPEEK on a variable after a CISET. The automatic display is never performed for variable procedures or class variables. CISETPEEK is only used when the input to *CI* is coming from the terminal. For command files, see CICMDFPEEK below.

CICMDFECHO
: Echo on the terminal each line that is read from a command file.

CICMDFPROMPT
: Echo the prompt (indented) on the terminal before each line read from a command file.

CICMDFEXIT
: Print an end-of-file message on the terminal after executing a command file.

CICMDFPEEK
: Display new value of a variable after assigning; used when input is from command file. See CISETPEEK, above.

FILES

/iu/tb/include/ci.h
/iu/tb/lib/sublib/cilib/*
/iu/tb/lib/sublib.a

SEE ALSO

ghough(1), phoenix(1), arglib(3), asklib(3), del(3), icp(3)

BUGS

The new push-level command is good, but perhaps an even simpler pause command would be useful during demos. "Push" just isn't very meaningful to most users. Pause might accept a print string as an argument, although this can be handled by echoing comment (:) statements (unless echoing is turned off). A separate print command might be useful for printing long paragraphs of information.

A stepping capability for automatically pausing after every command would be useful for running demos.

The asklib routines should exit on bad input from a batch file. We have fixed it so that EOF doesn't start an infinite loop, but we haven't prevented incorrect input from producing garbage runs.

The current help facility prints a redundant and annoying query or list of options when you ask for a nonexistent help file.

CI really should have entry and exit hooks so that arbitrary prologue and epilogue code could be run for each level invoked.

The "<" command should be extended to pass arguments to the script in the manner of UNIX shell files. (Note: In a sense CI and other command interpreters are simply alternate shells. It would make sense to share system code for these drivers.)

HISTORY

07-Feb-83 Laws at SRI-IU

Added the BUGS section and removed a few obsolete statements. Some of the source code has been restructured for integration with the Testbed, but very few capabilities have changed.

08-Oct-80 Steven Shafer (sas) at Carnegie-Mellon University
Added "class variables" and CICMDFPEEK.

12-Mar-80 Steven Shafer (sas) at Carnegie-Mellon University
CI now trims leading blanks and tabs from string variable assignments, and from the arg list for procedure variables and commands. The meta-help facility was fixed.

29-Jan-80 Steven Shafer (sas) at Carnegie-Mellon University
Created.

NAME

cmuimglib – CMU image access package

SYNOPSIS

#include "image.h"

IMAGE *imgopen(filename,mode)

IMAGE *imgcreat(filename,protection,pixbits,rstart,rend,cstart,cend)

imgclose(image)

IMAGE *imgdcreat(filename,protection,pixbits,rstart,rend,cstart,cend)

IMAGE *imgpopen(pathlist,filename,foundname,mode)

int imgscrap(image)

int pixel(image,row,column)

setpixel(image,row,column,value)

getrow(image,row,scol,ecol,vector)

putrow(image,row,scol,ecol,vector)

getcol(image,srow,erow,col,vector)

putcol(image,srow,erow,col,vector)

getbox(image,srow,erow,scol,ecol,array,dim)

putbox(image,srow,erow,scol,ecol,array,dim)

int imgshift(image,dr,dc,flg)

```
IMAGE *image;
char *filename,*pathlist,*foundname;
int mode,protection,pixbits,rstart,rend,cstart,cend;
int row,column,value;
int srow,erow,scol,ecol,dim;
int vector[],array[][dim];
int dr,dc,flg,size;
```

DESCRIPTION

This package provides mechanisms for emulating CMU image files manipulation. The routines are supported on the IU Testbed for compatibility with CMU, but the imglib package is recommended for new work.

Imgopen() takes the same parameters as open(2) does. It opens the named file and associates an IMAGE data structure with it. The PICPATH environment variable is used as a search path; it currently defaults to ":/iu/tb/pic:/aux/tbpic". Imgopen() returns a pointer to be used to identify the image in subsequent operations.

Imgcreat() creates both the IMAGE structure and the external file associated with an image. The filename and protection arguments are passed along to creat(2). Pixbits specifies the number of bits per pixel. Rstart, rend, cstart, and cend define the limits of the image matrix: the rows will run from rstart to rend and the columns will run from cstart to cend.

Imgclose() releases the resources (space and file descriptor) used by an image and ensures that the external file is up to date.

Imgdcreat() is just like imgcreat() except that it creates the directory if necessary. It returns 0 if unable to create the image file.

Imgpopen() is just like imgopen except that you may supply a search path. Use 0 for the path if you want the default: ":/iu/tb/pic:/aux/tbpic". The foundname argument should be the address of a buffer in which to put the full path name of the opened file; this argument cannot be omitted.

Imgscrap() is logically equivalent to imgclose() followed by unlink(), and thus gets rid of the image file. The return value is obtained from the call to unlink(). If your program creates a temporary image that need not be retained on disk, imgscrap() is the most efficient way to eradicate the image once it is no longer needed.

Pixel() returns the value of the pixel at coordinate (row,column) of the image. Setpixel() sets the pixel to "value".

Getrow() transfers the pixels on row "row" from column scol to ecol to the integer array "vector". Putrow() performs the opposite transfer.

Getcol() transfers the pixels on column "col" from row srow to erow to the integer array "vector". Putcol() performs the opposite transfer.

Getbox() and putbox() transfer a block of pixels. from an image to an int matrix, or from an int matrix to an image, respectively. Srow and erow specify the first and last rows of the image that are to take part; scol and ecol specify the first and last columns. "Array" is the address of a matrix of ints. "Dim" gives the number of elements physically allocated to each row of "array". (If srow == erow, then it doesn't matter what value you pass for dim.) Getbox() and putbox() are cover functions for getrow() and putrow().

Imgshift() shifts the row and column indices of an image. If the rows and columns run from rb to re and cb to ce, respectively then after the call to imgshift(), they will run from rb+dr to re+dr and cb+dc to ce+dc. However, if "flg" is 2 or 3 (i.e., bit 1 is set), the an absolute origin is set: the image will run from dr to dr+re-rb and from dc to dc+ce-cb. If "flg" is odd, the change is made permanent on the external file. [Permanent shifts are not supported on the Testbed.] Otherwise, the change is made only in the IMAGE structure but not in the corresponding IMAGEHEADER structure or in the external file. Imgshift() returns 0 for success.

An image is normally kept entirely in the process's address space, with vread(2) and vwrite(2) handling transfers between the disk file and virtual memory at the time of opening and closing the file. However, some images are too big for the virtual address space. If imgopen() or imgcreat() finds that it cannot "valloc" enough space for the image, then the image is set up to use software paging. This is transparent to the program, although performance is degraded (especially for pixel() and setpixel()). If software paging is found to be necessary, then the message "Warning: software paging for <image name>" is printed on the standard error channel.

IMAGE STRUCTURE

The cmuimglib routines can be used to read several picture formats and to write SRI TB01 picture formats; these are made to look like CMU picture files to the accessing code. The open picture files are manipulated using an IMAGE structure in the same way that Testbed routines use a PIC structure.

A CMU picture file contains a header page, a number of pages of pixels, and zero or more "trailer" pages. All pages are 1024 bytes long, and pixel pages consist of 32 rows of 32 bytes each. The number of pixels that will fit across a page is dependent on the physical pixel size. (The logical and physical pixel sizes can be different, but the *imgcreat* procedure makes them the same.) The pixel size may be any number from 1 to 32 bits. A pixel may cross a byte (or word or longword) boundary, but not a page boundary.

The format of the image header as contained on the external file is as follows:

```
#define IM_MAGIC          0x8281
#define IM_VERSION        1
#define IM_HDRPAGSIZ      1024L

typedef struct {
  short unsigned
      IM_magic,           Contains IM_MAGIC.
      IM_version;         Contains IM_VERSION.
  int
      IM_rows,            # of rows in image.
      IM_cols,            # of cols in image.
      IM_rstart,          Row # of upper left-hand pixel.
      IM_cstart,          Column address of leftmost column.
      IM_pixbits,         Logical # of bits per pixel.
      IM_physbits;        Physical # of bits per pixel.
  long
      IM_filhdrsiz;       Header + trailer size in bytes.
  char
      IM_propvallist;     Property/value list pointer.
} IMAGEHEADER;
```

The rest of the header (starting with IM_propvallist) and the trailer contain the list of property and value pairs (i.e., the property/value list may overflow the header page and grow indefinitely in the form of additional trailer pages). Properties and values are stored as null-terminated strings. They are simply stored sequentially: a property, then its value, then the next property, and so forth. A property followed by two (or more) nulls is considered to have no value, which is equivalent to having the null string as its value. There may be more than one null between a value and the subsequent property.

A copy of the file header is kept in main storage while an image is open, but it is NOT the same as the IMAGE structure that is addressed directly. Only the first (implementation-independent) part of the IMAGE structure is declared in "image.h." The full IMAGE structure is declared in the file "iimage.h", which is kept with the other sources of the image paging system. (Don't try to copy IMAGE structures.) The user-visible part of an IMAGE is as follows:

```
typedef struct {
  short unsigned
       IM_magic,                 Contains IM_MAGIC.
       IM_version;               Contains IM_VERSION.
  int
       IM_rows,                  # of rows in image.
       IM_cols,                  # of cols in image.
       IM_rstart,                Row # of upper left-hand pixel.
       IM_cstart,                Column address of leftmost column.
       IM_pixbits,               Logical # of bits per pixel.
       IM_physbits;              Physical # of bits per pixel.
  IMAGEHEADER *
       IM_header;                Pointer to copy of file header.
  int IM_rend,                   Highest row number in image.
       IM_cend;                  Highest column number in image.
  char *IM_filename;             File name of image.
} IMAGE;
```

It is not effective (and may be dangerous) for a user program to change values held in an open image object.

FILES

/iu/tb/include/image.h or /iu/tb/include/iimage.h
/iu/tb/lib/cmuimglib/*
/iu/tb/lib/cmuimglib.a
/iu/tb/lib/sublib.a
The -lm loader flag is also needed.

SEE ALSO

getprop(3), hdrlib(3), imagemac(3), imglib(3), piciolib(3), piclib(3), hdrformat(5)

DIAGNOSTICS

The CMU opening and closing routines are implemented on top of the Testbed piclib routines. These all use the printerr error reporting system, so you may get warning or error messages in that style. (You can also use the seterrflags(TRUE,TRUE,TRUE,TRUE) command to turn on execution tracing.)

Imgopen() returns zero if it can't open the file, or if the file is not an image file. Imgcreat() returns zero if it can't create the file or if its parameters are unbelievable.

Imgshift() returns -1 if "flg" is odd and image is not writable.

Other errors are fatal and provoke messages on the standard error output channel.

BUGS

The single IM_writeheader bit is used to trigger writing of both the header and any trailer or documentation record. The latter works fine, but writing the header currently causes an error message when the write(fd,...) statement fails for a valloc'ed picture file. Thus you get an extraneous error message if you ask for the IM_propvallist to be written on a hardware-paged image.

The implementation of imgopen() on top of the Testbed openimg() routine, and pixel() in parallel with the Testbed getpixel(), results in rather obscure code.

CMU image format is only one of many input formats supported on the Testbed for data conversion purposes. It is not currently supported for output.

The CMU IMAGEHEADER has provisions for property lists, but not for other types of user-supplied documentation. If this package is used to open non-CMU image formats, any associated documentation text is ignored. TB01 PROPDOC trailers are recognized and translated to IM_propvallist form, and the reverse conversion occurs during file closing. Reading of property lists from CMU-format images is not yet implemented, but can be provided if needed.

The IMAGE and IMAGEHEADER structures should be read-only from the point of view of the user of the image package.

HISTORY

26-Sep-83 Laws at SRI-IU
Added the getcol() and putcol() documentation.

26-Jan-83 Laws at SRI-IU
Imgpopen() is now [almost] redundant since imgopen() has been modified to search PICPATH. There seems to be no reason to maintain a non-searching version because you may obtain one by setting PICPATH to "" or ".".

17-May-82 Laws at SRI-IU
Implemented the cmuimglib opening and closing routines on top of the Testbed piclib routines, and the iolib routines in parallel with the Testbed piciolib routines.

21-Apr-81 Steve Clark (sjc) at Carnegie-Mellon University
Replaced IM_descriprion[IM_MXDESC] (which didn't work) with description of property/value stuff, which was just implemented.

15-May-80 Steve Clark (sjc) at Carnegie-Mellon University
Created imgpopen(). [Should really be imgopenp(). -- KIL]

19-Mar-80 David Smith (drs) at Carnegie-Mellon University
Clarified separation of IMAGE and IMAGEHEADER

27-Feb-80 David Smith (drs) at Carnegie-Mellon University
Getrow(), putrow() added.

07-Jan-80 David Smith (drs) at Carnegie-Mellon University
Created.

NAME

cmunmelib – CMU IMGNAME image name routines

SYNOPSIS

#include "imagename.h"

IMGNAME askimgname(prompt,gen,res,feat,ext,defalt)
```
char *prompt,*defalt;
int gen,res,feat,ext;
```

IMGNAME imgnamearg(cmdarg,gen,res,feat,ext,[askmsg])
```
int cmdarg;
int gen,res,feat,ext;
char *askmsg;
```

char *makeimg(iname,gen,res,feat,ext)
```
IMGNAME iname;
char *gen,*feat,*ext;
int res;
```

IMGNAME nextimgname(ptr,brk,prompt,gen,res,feat,ext,defalt)
```
char **ptr,*brk,*prompt,*defalt;
int gen,res,feat,ext;
```

IMGNAME parseimg(string)
```
char *string;
```

DESCRIPTION

This is a library of routines for manipulating CMU IMGNAME structures. It is currently supported on the Testbed for compatibility with existing CMU software, but the routines in imagelib/imgnmelib are preferred for new Testbed software.

Askimgname() prompts the user for an image name using the string *prompt*, with default *defalt*. The four int parameters *gen*, *res*, *feat*, and *ext* are flags indicating whether (nonzero) or not (zero) the calling program will substitute its own values for the corresponding fields in the resulting IMGNAME. If one of these parameters is nonzero and the user types a non-null value for the corresponding field, he is warned that it will be ignored and asked if he wishes to type in a new image name.

Makeimg() is the inverse function of *parseimg()*; it takes an IMGNAME and puts together a string that is a legal UNIX filename. The additional parameters *gen*, *res*, *feat*, and *ext*, if nonzero, are substituted for the corresponding fields given in the IMGNAME *iname*. Thus, if your program is to open several image files that are the same except for one or more fields that vary systematically, *iname* can be the template and *makeimg()* can be called once for each file desired, with the appropriate parameter equal to each instance of the varying field.

Nextimgname() is just like any of the nextXXX subroutines in parselib; it tries to read an image name from the string pointed to by *ptr;* if it can't, it calls *askimgname()* with the obvious arguments.

Parseimg() takes a string and parses an image name out of it. It returns an

IMGNAME structure. *Parseimg()* expects its string argument to be of this form:

<path>/*<genericimagename>*/*<resolution><feature>*.*<extension>*,

where *<path>* is a UNIX path, *<genericimagename>* is a UNIX filename, *<resolution>* is a string of digits, *<feature>* is a string of characters starting with a non-digit, and *<extension>* is a string of characters not containing a ".". If no path or no generic image name is given in the string, "." is assumed. If the resolution field is null, imgres is set to 0, and if the feature or extension is null, imgfeat or imgext is set to "".

EXAMPLE

Here is a program segment that will ask the user for an image name and open the red, green, and blue files of that image. The call to *askstring()* followed by the call to *parseimg()* should be replaced by a single call to the higher level subroutine *askimgname()*. The construction is shown here only for illustrative purposes.

```
#include "imagename.h"
#include "image.h"
#include "stringlib.h"
. . .

IMAGE *redimg, *greenimg, *blueimg;
IMGNAME iname;
char *filename, buffer[200];
int mode;
. . .

askstr("What (color) image?", buffer, "1.img");
iname = parseimg (buffer);

filename = makeimg (iname, 0, 0, "red", 0);
redimg = imgopen (filename, mode);

filename = makeimg (iname, 0, 0, "green", 0);
greenimg = imgopen (filename, mode);

filename = makeimg (iname, 0, 0, "blue", 0);
blueimg = imgopen (filename, mode);
. . .
```

IMGNAME STRUCTURE

Imagename.h contains the following definition of the IMGNAME structure:

```
        typedef struct {
                char imgpath[200];
                char imggen[20];
                int imgres;
                char imgfeat[15];
                char imgext[15];
        } IMGNAME;
```

The fields are used as follows:

imgpath
: a string containing the path of the generic image

imggen
: a string containing the generic image name, which is the directory of the various image files that come from one image

imgres
: the resolution of the image file--a resolution of n means that each pixel comes from n x n pixels in the original image

imgfeat
: a string containing the name of the feature in the image file

imgext
: a string containing the extension of the image file name.

EXAMPLE

This is a program fragment that shows the use of *askimgname()*, *nextimgname()*, *parseimg()*, *makeimg()*, *imgpopen()*, and *imgopen()*.

```
#include "imagename.h"
#include "image.h"
#include "stringlib.h"
. . .


IMAGE *densimg, *redimg, *greenimg, *blueimg;
IMGNAME iname;
char *argline, *filename, filefound[200];
. . .


iname = nextimgname (&argline, 0,
        "What r,g,b image?", 0, 0, 1, 0, "1.img");
filename = makeimg (iname, 0, 0, "red", 0);
redimg = imgpopen (0, filename, filefound, 0);
/* in order to be sure all three red, green, and blue image files come
   from the same place, we take filefound and use it to construct
   the green and blue image file names, and use imgopen (no search).
*/
iname = parseimg (filefound);
filename = makeimg (iname, 0, 0, "green", 0);
greenimg = imgopen (filename, 0);
filename = makeimg (iname, 0, 0, "blue", 0);
blueimg = imgopen (filename, 0);
filename = makeimg (iname, 0, 0, "dens", 0);
askstring ("What output image?", filename, filename);
/* We don't use askimgname here because we don't want to do any
   substituting afterwards--we constructed the usual default.
   Notice that filename still points to makeimg's static storage.
*/
densimg = imgcreat (filename, etc., etc.);
. . .
```

FILES

/iu/tb/include/imagename.h
/iu/tb/lib/cmuimglib/cmunmelib/*
/iu/tb/lib/cmuimglib.a

SEE ALSO

arglib(3), asklib(3), imgnmelib(3), parselib(3)

BUGS

Askimgnme() is also available under the earlier name of getimgnme(), and nextimgname() is also available as imgnmarg(). The current IMGNAME system is a mixture of original CMU contribution (which used IMAGENAME, get..., and ...arg) and the locally modified CMU system (which uses IMGNAME, ask..., and next...). Further emulation of the CMU system should revert to the original IMAGENAME software.

The IMGNAME returned by *parseimg()* and the string returned by *makeimg()* are in static storage, so each call destroys the result of the previous call.

HISTORY

18-Aug-83 Laws at SRI-IU
Changed the directory name from namelib to cmunmelib. Combined imgname(5) man page with this one.

10-Feb-83 Laws at SRI-IU
Created this man page from several others. Documented SRI modifications.

21-Jan-83 Laws at SRI-IU
A more sophisticated IMGNME name structure has been implemented in /iu/tb/lib/imglib/imgnmelib. It uses NULL and NOTINT instead of "." and 0 to mark missing fields, accepts a list of features instead of a single feature name, and is passed as a pointer instead of by structure copying. The IMGNAME structure documented here is now available only for the support of CMU code.

14-May-80 Steve Clark (sjc) at Carnegie-Mellon University
Created.

NAME

del – interrupt handling package

SYNOPSIS

```
#include "del.h"

ENABLEDEL;
IGNOREDEL;
DISABLEDEL;

...
DELBREAK;
...
...
DELRETURN;
...
...
DELRETN(value);
...
...
DELCLEAR;
...
```

DESCRIPTION

Del is a small subroutine used as an interrupt trap routine. It is used with the macro package <*del.h*> to provide a complete facility for trapping interrupts (*DEL* or *RUBOUT* key on the terminal).

The macro file incldes all the definitions listed above and uses an external int (_*del*_) as an interrupt flag. When an interrupt occurs, the *del* subroutine will increment _*del*_ (usually from 0 to 1), *signal*(2) itself for further interrupts, and return. The user may then test this variable with the macros listed above.

These macros control overall interrupt handling:

ENABLEDEL
: Clears _*del*_ and begins interrupt trapping.

IGNOREDEL
: Clears _*del*_ and ignores interrupts.

DISABLEDEL
: Clears _*del*_ and restores normal interrupt handling (kill process on interrupt).

These macros allow you to see if an interrupt has occurred (they do nothing if none has occurred):

DELBREAK
: Clears _*del*_ and acts like a *break* statement.

DELRETURN
: Clears _*del*_ and acts like a *return* statement.

DELRETN(value)
: Clears _*del*_ and acts like the statement *return*(*value*).

DELCLEAR
: Clears _*del*_ and prints "Break ignored".

In addition, you may perform a simpler test for interrupts as follows:

```
        if (_del_) ...
```

This is useful in conjunction with the above macros when you wish, for example, to exit from several layers of loops or procedures.

FILES

/iu/tb/include/del.h
/iu/tb/lib/sublib/errlib/del.c
/iu/tb/lib/sublib.a

SEE ALSO

signal(2)

DIAGNOSTICS

Break!

Printed by *DELBREAK*, *DELRETURN*, and *DELRETURN(value)* if an interrupt has occurred.

Break ignored.

Printed by *DELCLEAR* if an interrupt has occurred.

BUGS

It is possible, in any UNIX program that traps interrupts, for an interrupt to be missed if it immediately follows a previous interrupt of the same kind.

HISTORY

06-Dec-79 Steven Shafer (sas) at Carnegie-Mellon University
Created.

NAME

doclib – routines for manipulating documentation records

SYNOPSIS

#include "doclib.h"

Documentation structure (public fields only):

```
typedef DOC * struct {

   DOCTYPE doctype;          Documentation format code.
   string text;              Descriptive text.
   int textlen;              Length of text (except final null).

};
```

DOC closedoc(doc)
```
   DOC doc;
```

DOC copydoc(indoc)
```
   DOC indoc;
```

DOC newdoc([doctype,text,textlen]);
```
   DOCTYPE doctype;
   char *scantype;
```

DOC opendoc(filename,[offset])
```
   string filename;
```

printdoc(doc)
```
   DOC doc;
```

boolean validdoc(doc)
```
   DOC doc;
```

writedoc(filename,offset,doc)
```
   string filename;
   DOC doc;
```

DESCRIPTION

A DOC is the parsed representation of a documentation record. Such records are commonly used to store picture file descriptors or property lists. Currently implement documentation types simply store a text buffer and buffer length; no parsing is done. References to the fields should have the form "doc->text", where doc is declared to be of type DOC.

The doctype field may currently be NODOC (implying a null text string), TEXTDOC (with arbitrary character string), or PROPDOC (with a CMU-style property list).

Doclib subroutines:

closedoc(doc) -> DOC
: If the DOC is multiply linked, simply decrement the link count; otherwise free the DOC structure. The return value is generally NULL, but may be the original DOC value if an error occurred.

copydoc(indoc) -> DOC
: Copy all data associated with indoc to a new DOC. The two DOCs will then be entirely separate.

newdoc([doctype,text,textlen]);
: Create a new DOC with the specified format, text, and text buffer length. The defaults are TEXTDOC format, a user query for the text string, and the length of the resulting string. If the NODOC format is used, the default text is a null string.

opendoc(filename,[offset]) -> DOC
: Open the specified file and parse the documentation record. The record must currently have the Testbed IMGDOC format defined for image file trailer records; any other format will cause a NODOC structure to be returned.

printdoc(doc)
: Print the text associated with a DOC structure. This routine is not currently able to print PROPDOC text.

validdoc(doc) -> boolean
: Check for a non-null DOC with a valid structure identification code.

writedoc(filename,offset,doc)
: Write a documentation record at the specified file address.

EXAMPLES

See describe.c and the document.c and copypic.c routines in /iu/tb/lib/imagelib/demo for examples of use.

FILES

/iu/tb/include/doclib.h
/iu/tb/lib/sublib/doclib/*
/iu/tb/lib/sublib.a

SEE ALSO

blklib(3), getprop(3), piclib(3), docformat(5)

DIAGNOSTICS

The doclib routines use the printerr error reporting system. An error will typically cause a message to be printed on the error stream and the routine will return a null DOC.

BUGS

There is no hook for specifying other than an IMGDOC format. In particular, there is no provision for reading straight text that is not introduced by an IMGDOC header. This could be permitted if an optional text length argument were added to opendoc().

Writedoc() will garbage your file if you give it an incorrect offset, so be very careful. This suggests that using trailer records for documentation is not particularly wise. The need for write access may also lead to situations where other routines can destroy the file.

Opendoc() and writedoc() take filenames instead of file descriptors as arguments. In the case of writedoc(), this was necessary because writes to a valloc'ed file descriptor (e.g., a pic->fd) did not work. It also allows writedoc() to work for a picture that was not opened for write access. It could create problems, however, in writing documentation to a file version that has not yet been closed and written out to disk.

See the source file headers for additional comments.

HISTORY

06-Sep-83 Laws at SRI-IU
Created.

NAME

dsplib – device-independent display allocation

SYNOPSIS

#include "dsplib.h"

Display descriptor (public fields only):

```
typedef DSP * struct {
   string devname;            Print name for the device.
   int nmems;                 Number of memory planes.
   int membits;               Bit depth of each memory.
   int memcols;               Number of columns per memory.
   int memrows;               Number of rows per memory.
};
```

DSP closedsp(dsp)
DSP dsp;

DSP copydsp(indsp)
DSP indsp;

DSP linkdsp(indsp)
DSP indsp;

DSP opendsp(devname)
string devname;

printdsp(dsp)
DSP dsp;

testdsp(testnbr, dsp)
DSP dsp;

boolean validdsp(dsp)
DSP dsp;

DESCRIPTION

A DSP is the handle (or "capability") by which a physical display device is accessed. Public fields are listed above. They should usually be treated as read-only. References to the fields should have the form "dsp->memcols," where dsp is declared to be of type DSP.

closedsp(dsp) -> DSP

If the DSP is multiply linked, simply decrement the link count. Otherwise close the display device and free the DSP structure. The return value is generally NULL, but may be the original DSP value if an error occurred.

copydsp(indsp) -> DSP

Copy the device parameters associated with indsp to a new DSP. The two DSPs will then be entirely separate except for their side effects on a single physical device. Closing either DSP may make the other unusable.

linkdsp(indsp) -> DSP
: Copy the original DSP pointer and increment the link count of the associated data structure. Closing either DSP will simply break the link.

opendsp(devname) -> DSP
: Open the specified display device. The only legal values for devname at present are "grinnell" and NULL (which is a default for "grinnell").

printdsp(dsp)
: Print partial contents of a DSP structure. This is primarily used for debugging.

testdsp(testnbr,dsp)
: Run diagnostic tests on the display. You may specify tests one through four on the Grinnell, or test zero to sequence through all four.

validdsp(dsp) -> boolean
: Check for a non-null DSP with a valid structure identification code.

FILES

/iu/tb/include/dsplib.h
/iu/tb/lib/imagelib/dsplib/*
/iu/tb/lib/imagelib.a
/iu/tb/lib/sublib.a

SEE ALSO

dpy(1), imgsys(1), blklib(3), frmlib(3), gkeycur(3), gmrlib(3), imglib(3)

DIAGNOSTICS

The dsplib routines use the printerr error reporting system. An error will typically cause a message to be printed on the error stream and the routine will return a null DSP. If given a null DSP as an argument, the opening routines will generally just open the default device.

BUGS

The only device currently available is the Grinnell. You get it regardless of the device name you pass to opendsp().

The screen flicker that results from initializing lookup tables cannot be suppressed by turning the CRT off during the initialization. The problem seems to involve the self-test board, which is invoked to reset the lookup tables. The alternative is to avoid this call and do the initialization step by step.

It probably makes more sense to link to a DSP than to copy one, but both capabilities will be provided until this is determined.

See the source directory for additional comments.

HISTORY

24-Nov-82 Laws at SRI-IU
: Created.

NAME

dynload – dynamically load object modules

SYNOPSIS

Dynamic_Load(object_file,symbol_file,"sym1",&s1,...,0);

DESCRIPTION

The argument list ends with an arbitrary number of symbol/pointer pairs terminated by a zero. The fields are as follows:

object_file
: Filename of the object file to be dynamically loaded into the running program.

symbol_file
: Filename for the currently running program (to be used as a symbol table for the link step).

"Sym1"->"SymN"
: Symbols that the user would like address for when the dynamic load is finished. These symbol names must usually start with an underscore.

&s1->&s1
: Pointers to variables in which to place the values for the specified symbols.

Dynamic_Load("sub.o","main","_Routine",&R,"_Var",&V,0) will return variable R with the address of Routine and V with the address of Var.

EXAMPLE

Compile and run this file, "test.c":

```
main()
{
 int (*entry_pt)();
 char **rname;

 if (Dynamic_Load("sub.o","test","routine",&entry_pt,
    "Routine_Name",&rname,0) < 0) exit(0);
 printf("Routine %s is at %x.0,*rname,entry_pt);
 if (entry_pt == 0) exit(0);

 printf("Calling entry_pt.0);
 (*entry_pt)(10);
}

foo(p) {printf("  Foo called with p = %d.0,p);}
```

to load and execute file "sub.o" compiled from:

```
char *Routine_Name = "Test Routine";

routine(arg)
{
 printf("Calling foo(%d).0,arg);
```

```
    foo(arg);
  }
```

Note that the test routine could also have compiled the subroutine dynamically using the UNIX exec() or system() routines.

FILES

/iu/tb/lib/sublib/syslib/dynload.c /iu/tb/lib/sublib.a

SEE ALSO

ld(1), Franz Lisp (cfasl)

DIAGNOSTICS

Dynload returns a 0 for success, -1 for failure.

BUGS

Currently restricted to use with UNIX object files.

HISTORY

04-Aug-83 Kashtan at SRI-IU
Created.

NAME

editor — execute the user's favorite editor

SYNOPSIS

```
int editor(file,message);
  char *file,*message;
```

DESCRIPTION

Editor will execute the editor preferred by the user, to edit the file whose name is *file*. The string *message* will be printed to tell the user what he is editing.

This routine contains all the special knowledge needed to execute the editors known to the system; if the user wants to use some other editor, it will be used, but with no special assistance from this routine.

There is assumed to be an environment parameter called *EDITOR*, whose value is the name of the user's favorite editor (e.g., *ex*, *emacs*). If the *EDITOR* parameter is missing from the environment, the default editor (currently "vi") will be used.

The *runp*(3) routine is used to execute the editor; it uses the *PATH* environment parameter to find the editor. In general, if the editor can be found by the shell, it should be found by *editor*.

Editor returns the return code of the editor upon normal completion, or -1 if the editor is interrupted or cannot be executed.

This routine is useful for any program that allows the user to edit a text file.

ENVIRONMENT

EDITOR
: is assumed to be the name of the user's favorite editor. The editors currently in use are: "ex", "vi", "emacs", and "ed". Any other editor is also acceptable; it will be executed with a single parameter—the name of the file to be edited. The value of EDITOR may be an absolute name, if desired.

PATH
: is assumed to be the searchlist for executable programs.

FILES

/iu/tb/lib/sublib/syslib/editor.c
/iu/tb/lib/sublib.a

SEE ALSO

ed(1), emacs(1), ex(1), runp(3)

DIAGNOSTICS

Returns -1 if the editor cannot be successfully executed, 0 normally, >0 if the editor itself discovers some error condition.

BUGS

Currently configured for the editors in use at CMU and on the Testbed.

If *message* contains a new line, it may not be completely meaningful when using some screen-oriented editors.

HISTORY

12-Nov-80 Mike Accetta (mja) at Carnegie-Mellon University
Removed references to obsolete editors.

06-Dec-79 Steven Shafer (sas) at Carnegie-Mellon University
Created.

NAME

filelib — file manipulation routines

SYNOPSIS

#include <stdio.h>

int filecopy(here,there)

int ffilecopy(here,there)
   FILE *here,*there;

FILE *fopenp (searchlist,filename,buffer,mode)
   char *searchlist,*filename,*buffer,*mode;

FILE *fwantread (searchlist,filename,fullname,prompt)
   char *searchlist,*filename,*fullname,*prompt;

FILE *fwantwrite (searchlist,filename,fullname,prompt,warn)
   char *searchlist,*filename,*fullname,*prompt;

boolean isafile(filename)
   char *filename;

int openp (searchlist,filename,buffer,mode)
   char *searchlist,*filename,*buffer;

pclose(stream)
   FILE *stream;

FILE *popen(command, type)
   char *command, *type;

int printfile(filename)
   char *filename;

int rename (oldname,newname)
   char *oldname,*newname;

int wantread (searchlist,filename,fullname,prompt)
   char *searchlist,*filename,*fullname,*prompt;

int wantwrite (searchlist,filename,fullname,prompt,warn)
   char *searchlist,*filename,*fullname,*prompt;

DESCRIPTION

Filecopy copies the open input file *here* to the open output file *there*. The input data from the current file pointer of *here* until EOF are appended after the current file pointer of *there*.

Ffilecopy copies the open buffered input file *here* to the open buffered output file *there*. The input data from the current file pointer of *here* until EOF are appended after the current file pointer of *there*. The data currently in both buffers is correctly handled, and both buffers are left in an appropriate state. After *ffilecopy*, *feof(here)* will be true.

These routines are probably faster than you would care to code for yourself, although they are quite simple.

Isafile is a simple boolean function that returns true if the file exists and false if it does not.

Openp and *fopenp* use *searchp* to search for a file and open the file when and if it is found. The value returned will be the normal value of *open*(2) or *fopen*(3s) -- a file descriptor or FILE pointer on success, and a -1 or a 0 on failure.

Searchlist must be a list of directory names separated by colons; one by one, these names are parsed and concatenated (with a separating slash) onto *filename* to form a complete pathname. An attempt is then made to open the file with this name; if successful, *openp* and *fopenp* return with a success indication; otherwise, searching continues. If success is achieved, then the resulting pathname is copied into the string *buffer* provided by the user. The *open* and *fopen* calls are made with the appropriate *mode* as specified by the parameter.

The arguments to *popen* are pointers to null-terminated strings containing a shell command line and an I/O mode, either "r" for reading or "w" for writing. It creates a pipe between the calling process and the command to be executed. The value returned is a stream pointer that can be used (as appropriate) to write to the standard input of the command or read from its standard output.

A stream opened by *popen* should be closed by *pclose*, which waits for the associated process to terminate and returns the exit status of the command.

Because open files are shared, a type "r" command may be used as an input filter, and a type "w" as an output filter.

Rename attempts to change the name of the file *oldname* to *newname*. *Rename* uses *link*(2) and *unlink*(2) if possible, since this is the fastest way to change the name of a file. Under certain circumstances (i.e., if *oldname* and *newname* specify different mounted devices), this may fail; in such a case, *rename* will *creat*(2) the result file and use *filecopy*(3) to copy the data.

If *rename* is successful, then *oldname* will be deleted and *rename* will return a 0. If unsuccessful, *newname* will be deleted (if any file already has this name), and -1 will be returned.

Searchp looks for an acceptable filename by concatenating a name onto each directory name within a given "search list."

Searchlist is a list of directory names, separated by colons (:). *Searchp* will parse these names, prefixing each in turn to *filename*, the name of the file being sought. The resulting pathname is passed as an argument to *function*, a function provided by the user. This function will receive one parameter -- the pathname -- and must return an integer telling whether this filename is acceptable or not. If a nonzero value is returned, then the search continues with the next directory name from *searchlist*. If the value 0 is returned, then searching stops. In this case, the full filename is copied into the string *buffer*, and *searchp* returns 0.

If all directories are unsuccessfully searched, then *searchp* returns the value -1.

If *filename* begins with a slash, it is assumed to be an absolute pathname and *searchlist* is not used.

Wantread attempts to open a file for input, asking the user for an alternate file name over and over again until a file is successfully opened for reading.

Searchlist is a list of directories that may contain the file, with the directory names separated by colons. *Filename* is the name of the file the program wishes to open; if *filename* is the null string, then the user will be asked immediately for the name of a file to open. When a file is opened, the complete filename is copied into *fullname*, which must be a string provided by the user.

If *openp*(3) fails to open the desired file, then an error message is printed along with the message *prompt*, and the user can type in an alternate file name. The new file name is searched for using the same *searchlist*. The user may also indicate that no file is acceptable; in this case, *wantread* will return an error indication.

Fwantread is the same as *wantread*, but uses *fopenp*(3) to open a buffered file, and returns a FILE pointer.

Wantwrite attempts to open a file for output, asking the user for an alternative file name over and over again until a file is successfully opened for writing.

Searchlist is a list of directories which may be contain the file, with the directory names separated by colons. *Filename* is the name of the file the program wishes to open; if *filename* is thenull string, then the user will be asked immediately for the name of a file to open. When a file is opened, the complete filename is copied into *fullname*, which must be a string provided by the user.

For each directory whose name appears in *searchlist*, the complete filename will be formed by contatenating a slash (/) and *filename*. Then, an attempt will be made to *creat*(2) the file. If *warn* is TRUE, then a check will first be made to ensure that the file does not already exist. If it does, the user will be asked if he wants to delete it; if he says "no," then no attempt will be made to *creat* the file and searching will continue with the next directory.

If no attempt to *open*(2) the file is successful, then an error message is printed along with the message *prompt*, and the user can type in an alternative file name. The new file name is searched for using the same *searchlist*. The user may also indicate that no file is acceptable; in this case, *wantwrite* will return an error indication.

Fwantwrite is the same as *wantwrite*, but uses *fopen*(3) to create a buffered file, and returns a FILE pointer.

DIAGNOSTICS

Wantwrite returns -1 on error; otherwise the file descriptor of the successfully created file.

Fwantwrite returns 0 on error, or the FILE pointer of the successfully created file.

Printfile simply prints the named file.

FILES

/iu/tb/lib/sublib/filelib/*
/iu/tb/lib/sublib.a

SEE ALSO

syslib(3)

DIAGNOSTICS

Filecopy and *ffilecopy* return 0 normally, -1 on error. If an error occurs, the file copying operation may be incomplete.

Openp returns -1 on error (no openable file found); *fopenp* returns 0.

Popen returns a null pointer if files or processes cannot be created, or the Shell cannot be accessed.

Pclose returns −1 if *stream* is not associated with a 'popened' command.

Rename returns 0 normally, -1 on error. If an error occurs, *oldname* still exists.

Searchp returns -1 if no filename is satisfactory, 0 otherwise.

Wantread returns -1 on error, otherwise, the file descriptor of the successfully opened file.

Fwantread returns 0 on error, or the FILE pointer of the successfully fopened file.

SEE ALSO

open(2), mv(1), link(2), unlink(2), creat(2), filecopy(3), pipe(2), fopen(3), fclose(3)

BUGS

Buffered reading before opening an input filter may leave the standard input of that filter mispositioned. Similar problems with an output filter may be fore−stalled by careful buffer flushing, e.g., with *fflush*−see *fclose*(3).

UNIX from Bell Labs uses three different searching algorithms in three different contexts. In the shell, executable programs are sought, but are considered absolute pathnames if they *contain* a slash, even if it is not the first character (bogus, in my opinion − sas). The routines execvp and execlp use the same rule, but also accept the minus sign (-) as a separator in the pathlist. The C compiler, in searching for macro files, uses the rule of *searchp* − a filename is only absolute if it *begins* with a slash.

Users normally do not need to use *searchp;* there are other, higher level routines (*runp*, *runvp*, *openp*, *fopenp*) that should normally be used. There are, however, occasions in which such routines are not powerful enough; then, *searchp* is appropriate.

Fullname must be long enough to hold the complete filename of the opened file.

There should be a way to change the value of *searchlist* if the search for a file is unsuccessful.

HISTORY

11-Feb-83 Laws at SRI-IU
    Combined the separate **man** pages. Added the printfile routine.

06-Dec-79 Steven Shafer (sas) at Carnegie-Mellon University
    Modified for CMU – popen now performs path search for shell, instead of directly executing "/bin/sh".

NAME

frmlib – device-independent virtual display manipulation

SYNOPSIS

#include "frmlib.h"

Frame descriptor (public fields only):

```
typedef FRM * struct {
    DSP dsp;                  Display device.
    FRMTYPE frmtype;          Testbed frame type code.
    int ncols;                Maximum number of columns.
    int nrows;                Maximum number of rows.
    int pixelbits;            Bits per pixel.
    int cmufrm;               CMU frame number.
    int cmufrmtype;           CMU frame type code (mode).
};
```

FRM closefrm(frm)
  FRM frm;

colorfrmwdw(frm,wdw,value0,value1,...)
  FRM frm;
  WDW wdw;

FRM copyfrm(infrm)
  FRM infrm;

erasefrm(frm)
  FRM frm;

erasefrmwdw(frm,wdw)
  FRM frm;
  WDW wdw;

FRM linkfrm(infrm)
  FRM infrm;

FRM openfrm([dsp,frmtype,pixelbits])
  DSP dsp;
  FRMTYPE imgtype;

FRM openfrm(devname)
  string devname;

printfrm(frm)
  FRM frm;

showfrm(frm)
  FRM frm;

boolean validfrm(frm)

```
FRM frm;
```

DESCRIPTION

A FRM is the handle (or "capability") by which portions of a physical display device are accessed. Public fields are listed above. They should usually be treated as read-only. References to the fields should have the form "frm->ncols," where frm is declared to be of type FRM.

The current implementation is an attempt to rationalize the CMU frame concept and extend it to devices other than the Grinnell. Unfortunately this process has not proceeded very far, and the user may need to use the underlying CMU frame structure to communicate with the CMU Grinnell code.

The conceptual basis of a FRM is a virtual display device composed of one or more image memories with associated display parameters (look up table contents, color gun assignments, cursors, etc.). The FRM may or may not be visible, depending on the most recent call to showfrm(). The various display parameters may or may not be controllable after the FRM is opened; see the CMU Grinnell software documentation.

Frame type codes for the FRMBLK structure.

```
typedef enum {
  NOFRM,                  Undefined image type.
  REDOVL,                 Red overlay.
  GREENOVL,               Green overlay.
  BLUEOVL,                Blue overlay.
  WHITEOVL,               White overlay.
  BWFRM,                  BW (monochrome) image.
  RGFRM,                  Anaglyphic stereo pair.
  RGBFRM                  RGB color image.
} FRMTYPE;
```

Frmlib routines:

closefrm(frm) -> FRM

If the FRM is multiply linked, simply decrement the link count. Otherwise close the CMU frame and free the FRM structure. (The display device is not closed.) The return value is generally NULL, but may be the original FRM value if an error occurred.

colorfrmwdw(frm,wdw,value0,value1,...)

Set the indicated window to a constant brightness or color. One value should be supplied for each memory plane in the frame type.

copyfrm(infrm) -> FRM

Copy the original frame fields to a new FRM. The two FRMs will then be entirely separate except for their side effects on a single CMU frame system. Closing either FRM may make the other unusable.

erasefrm(frm)
: Erase (set to zero) the display memory associated with the FRM.

erasefrmwdw(frm,wdw)
: Erase the indicated window into the frame memory.

linkfrm(infrm) -> FRM
: Copy the original FRM pointer and increment the link count of the associated data structure. Closing either FRM will simply break the link.

openfrm([dsp,frmtype,pixelbits]) -> FRM
: Obtain a frame on the specified display device. The defaults are "grinnell", BWFRM, and 1 bit for an overlay or dsp->membits for a picture frame. In case of failure, the returned value will be NULL and the global error code will be ERROR [see printerr(3)].

printfrm(frm)
: Print partial contents of a FRM structure. This is primarily used for debugging.

showfrm(frm)
: Display the FRM. This configures the display device to the frame's display parameters.

validfrm(frm) -> boolean
: Check for a non-null FRM with a valid structure identification code.

FILES

/iu/tb/include/frmlib.h
/iu/tb/lib/imagelib/frmlib/*
/iu/tb/lib/imagelib.a
/iu/tb/lib/sublib.a

SEE ALSO

blklib(3), dsplib(3), imgfrmlib(3), imglib(3), piclib(3)

DIAGNOSTICS

The frmlib routines use the printerr error reporting system. An error will typically cause a message to be printed on the error stream and the routine will return a null FRM. If given a null FRM as an argument, the opening routines will generally ask for the necessary information.

BUGS

It probably makes more sense to link to a FRM than to copy one, but both capabilities will be provided until this is determined. Linkfrm() does not currently increment the link count of the contained DSP.

See the source directory for additional comments.

HISTORY

26-Nov-82 Laws at SRI-IU
: Changed setfrmwdw() to colorfrmwdw() and ersfrmwdw() to erasefrmwdw(), made openfrm() arguments optional.

24-Nov-82 Laws at SRI-IU
Created.

NAME

getprop, putprop, delprop, getproplist – CMU image property list routines

SYNOPSIS

int getprop(image,property,value);

int putprop(image,property,value);

delprop(image,property);

int getproplist(image,&properties,&values);

```
IMAGE *image;
char property[],value[],**properties,**values;
```

DESCRIPTION

These routines are only implemented for the cmulib image access functions, not the piclib and imglib access recommended for general testbed use.

These subroutines allow you to manipulate property/value pairs in image file headers. There are two kinds of uses of properties: ones that have values associated with them (such as *description*), and ones that are either present or absent (such as *signed*). Both types may be manipulated easily by this package.

Properties and values are null-terminated strings; they may consist of any characters except the null character (0 byte). A property with no value is the same as a property with the null string as its value. If **property** is zero or if it is the null string, **getprop** and **putprop** will return failure. The return values of these subroutines are boolean indications of failure, so you can say

if (foo) deal with error;
continue

Getprop returns 0 (false) if the given property is in the image's header and 1 (true) if not. If the property is not present it does not touch **value.** If **value** is nonzero, then the value of **property** is copied into it. To facilitate access to properties that you don't expect to have values, if **value** is zero, no attempt is made to copy the value of **property.**

Putprop returns 0 (false) if it successfully adds **property** and **value,** and 1 (true) if it does not. It will succeed unless it is given bogus arguments; the image header will grow to accommodate an "infinite" number of properties and values. If **property** is already in the list, its value is changed to **value;** otherwise both **property** and **value** are added. **Value** may be 0, in which case **property** is given the null value.

Delprop deletes the property and its value from the image header. It does not return a value; therefore you cannot tell if **property** actually was in the image header.

Getproplist returns 0 (false) if it succeeds and 1 (true) if it fails to get a list of the properites and values in an image's header. Upon successful return, **properties** and **values** point to zero-terminated arrays of pointers to strings, suitable for passing to *prstab(3)* or *stablk(3)*, for example. The strings that they point to are the original copies in the image header, so they must not be altered. The arrays of pointers are *malloc(3)* 'ed by **getproplist**, so it is appropriate for you to *free* them.

FILES

/iu/tb/lib/cmuimglib/cmupiclib/prop.c

SEE ALSO

cmuimglib(3), doclib(3), and CMU IUS Whitepaper CMU003, "Image Access Routines."

DIAGNOSTICS

These subroutines return 1 (true) for a failure.

BUGS

The documentation may grow but it will never shrink. The minimum length is about 1016 bytes.

The proplist routines are currently implemented only for TB01-header picture files. Conversion of proplist data from the CMU format will be added if it is needed.

HISTORY

10-Apr-81 Steve Clark (sjc) at Carnegie-Mellon University
Created.

NAME

getwd – get working directory

SYNOPSIS

getwd (dir);
  char *dir;

DESCRIPTION

Getwd places the pathname of the current working directory into the string *dir*.

FILES

/iu/tb/lib/sublib/syslib/getwd.c
/iu/tb/lib/sublib.a

SEE ALSO

pwd(1), chdir(1), path(3), expand(3)

DIAGNOSTICS

Returns -1 on error (directory unreadable or unsearchable), 0 otherwise.

BUGS

The pathname cannot be more than 120 characters long.

Since the current directory is changed during the process, an error causes the current directory to be (effectively) undefined.

HISTORY

05-Dec-79 Steven Shafer (sas) at Carnegie-Mellon University
  Created.

NAME

gmrfrmlib – CMU Grinnell frame manipulation routines

SYNOPSIS

#include "grinnell.h"

g_init()

g_exit()

g_allocfrm(mode,pixelsize)
int mode,pixelsize;

g_freefrm(frm)
int frm;

g_splitfrm(frm,bits,newfrm)
int frm,bits,*newfrm;

DESCRIPTION

G_init() initializes the Grinnell display package. It should be called before any other routines in the package.

G_exit() deallocates all frames and realeases the Grinnell. G_init() must be called again after g_exit() if any library routines are to be used. It is not necessary to call g_exit() before exiting a program.

G_allocfrm() allocates a piece of the Grinnell memory. Mode may be any one of the following:

G_BLUEOVERLAY
G_GREENOVERLAY
G_REDOVERLAY
G_WHITEOVERLAY
G_MAPPED
G_RGB
G_BW
G_OVERLAY.

If mode is G_OVERLAY, the first available overlay will be allocated. If mode is G_RGB, the byte size of each band will be pixelsize. The allocation algorithm looks for the smallest space available to set up the frame so as to leave large pieces free if they are needed.

G_freefrm() frees a frame that has been allocated. The memory is then available for future allocation.

G_splitfrm() makes two frames out of one. "Frm" is the frame to split. "Bits" is the new byte size for frm. "Newfrm" is the address of the new frame identifier for return. After the splitting operation is performed, newfrm will contain the most significant, and frm will contain the least significant bits of the old frm. The frame may be of any type.

FILES

/iu/tb/include/grinnell.h
/iu/tb/lib/imagelib/gmrlib/frmlib/*

/iu/tb/lib/imagelib.a

SEE ALSO

frmlib(3), gmrlib(3)

DIAGNOSTICS

All routines return -1 on error.

BUGS

It is not possible to merge a frame that has been split.

HISTORY

25-Aug-80 Jerry Denlinger (jld) at Carnegie-Mellon University
Created.

NAME

gmrlib – Grinnell graphics library

SYNOPSIS

crslib

crsarg(cmdarg,limitwdw,dftpnt,[askmsg]) -> PNT
g_getcur(frm,curnum,row,col,visible,blinking)
g_keycur(curnum,lpr0,srow,scol,erow,ecol,argflg,usrfnc)
g_setcur(frm,curnum,row,col,visible,blinking)
g_setsumscale(scale,xoff,yoff)
g_tabcur(curnum,lpr0,srow,scol,erow,ecol,argflg,g_funct)
g_trackcur(frm,curnum,erow,ecol,trkdev,toggle)
ttyreset()

ctllib

g_chgdisp(frm)
g_ersfrm(frm)
g_gmroff()
g_gmron()
g_ifvc()
g_intest(tnum)
g_ipc()
g_setdebug()
g_setstate()
g_size()

frmlib

g_allocfrm(mode,pixelsize) -> int
g_atom(s)
g_cmputmsk(frd,index)
g_exit()
g_findbits(psiz)
g_freefrm(frd)
g_frmtype(frm) -> int
g_getcont()
g_getframe()
g_getmode(prompt,deflt) -> int
g_init(selected_monitor) -> int
g_ldfdflt(frd,mode)
g_mtoa(d,b)
g_releaseotherifvcs()
g_retrifvc(fread)
g_saveifvc(fwrite)
g_seizeotherifvcs() -> int
g_splitfrm(frm,bits,newfrm)
g_validfrm(frm)

gmrlib/blkpixel.c

g_blkcget(frd,srow,erow,scol,ecol,rbuf,gbuf,bbuf)
g_blkcset(frd,srow,erow,scol,ecol,rbuf,gbuf,bbuf)
g_blkget(frm,srow,erow,scol,ecol,buffer)
g_blkmov(sfrd,srow,erow,scol,ecol,dfrd,dsrow,dscol)
g_blkpnts(npnt,odd,srow,erow,scol,ecol)
g_blkset(frm,srow,erow,scol,ecol,inbuf)
g_fixup(buffer,wrdcnt)
g_rectread(fd,srow,erow,scol,ecol,inbuf) -> int
g_rectwrite(fd,srow,erow,scol,ecol,outbuf)
g_sbbfd(fd,inbuf,srow,erow,scol,ecol,outbuf,wrdcnt,oddbyte)
g_sbpfd(fd,srow,erow,scol,ecol,pval,buffer,bufcnt)
g_setblkpix(frm,srow,erow,scol,ecol,pval)
g_setcblkpix(frd,srow,erow,scol,ecol,rv,gv,bv)

gmrlib/clip.c

code(x,y,mbrx1,mbrx2,mbry1,mbry2)
g_lineclip(srow,scol,erow,ecol,mbrsr,mbrsc,mbrer,mbrec)
g_listclip(frm,buffer,mbrsr,mbrsc,mbrer,mbrec,h,w)

gmrlib/digitize.c

g_vdigoff()

gmrlib/dumpfp.c

g_dmpfp(f)

gmrlib/gmrcursor.c

g_joycur(curnum,lpr0,srow,scol,erow,ecol,flag,g_funct)
g_rdcur(curnum,row,col,ef,ff)
g_wrtcur(curnum,row,col,lpr0)

gmrlib/gpixel.c

g_getcpixel(frm,row,col,rv,gv,bv) -> int
g_getpixel(frm,row,col,pval) -> int
g_gfd(frm,frd,fdex)
g_setcpixel(frm,row,col,rv,gv,bv) -> int
g_setpixel(frm,row,col,pval) -> int

gmrlib/graphic.c

g_assocfrm(ifrm,gfrm)
g_deassocfrm(ifrm,gfrm)
g_getdsplst(g_addr,roff,coff,npnt)
g_goverlay(frm,srow,scol,erow,ecol,g_height,g_width,g_vr,g_we)
g_linedraw(frm,srow,scol,erow,ecol,g_height,g_width)
g_linerase(frm,srow,scol,erow,ecol,g_height,g_width)
g_listdraw(frm,g_displist,g_height,g_width)
g_listerase(frm,g_displist,g_height,g_width)
g_ovloff(ifrm,gfrm)
g_ovlon(ifrm,gfrm)
g_rectdraw(frm,srow,erow,scol,ecol)
g_recterase(frm,srow,erow,scol,ecol)
g_setdsplst(g_addr,roff,coff,npnt)
g_veclist(frm,g_dsplist,g_height,g_width,g_we)

gmrlib/io.c

g_flushbuff() -> int
g_inspect(addr,bytes,istr)
g_ioctl(nocnt,inspect)
g_multiioctl(command,dataaddr) -> int
g_plinesem()
g_read(buffer,nbytes,inspect) -> int
g_setbuf(bufflg)
g_vlinesem()
g_write(addr,bytes,inspect) -> int

gmrlib/map.c

g_fillmap(g_buf,bsize,sbit,g_mval)
g_genmap(g_buf,bsize,sbit)
g_mapaddr(frm,mapaddr)
g_mapcore(mapfile,mbuf)
g_mapfile(frm,mapfile) -> int
g_mapwrite(frd,mode)
g_mbulkload(frm)
g_retrbulkmap(fread)
g_savebulkmap(fwrite)

gmrlib/packer.c

g_blkdecode(g_ibuf,nwrd,fbs,fsb,g_obuf)
g_blkencode(g_ibuf,nwrd,fbs,fsb,g_obuf)
g_decode(g_wrd,fbs,fsb,g_val)
g_encode(g_wrd,fbs,fsb,g_val)
g_locpack(row,column,g_addr,nxtpntr)
g_recpack(srow,erow,scol,ecol,g_addr,nxtpntr)

gmrlib/savestate.c

g_retrstate(fname)
g_savestate(fname)

gmrlib/scncnvrt.c

g_scncnvrt(frame,intensity,nvertices,vertexlist)
g_cscncnvrt(rgbframe,red,green,blue,nvertices,vertexlist)

struct pnt { int xpnt; int ypnt;};
struct pnt vertexlist[];

gmrlib/string.c

g_str(frm,str,row,col,g_height,g_width,dir,noboxes,drawstr)
g_strdraw(frm,str,row,col,g_height,g_width,dir)
g_strerase(frm,str,row,col,g_height,g_width,dir)

gmrlib/summa.c

g_rdsumma(xpos,ypos,pen,flag1,flag2,flag3)
g_sumclose()
g_suminit()
g_tsumread()

gmrlib/vdig.c

g_vdig(frame,shift,cont_mode,threshold,function,sumshf)
g_vdoff()
g_vdon()

gmrlib/vdsetup.c
gmrlib/vdsnap.c

g_vdsetup(frame,threshold,prompt) -> int
g_vdsnap(frame,threshold)

gmrlib/vline.c

g_vclinedraw(frm,srow,scol,erow,ecol,height,width,rv,gv,bv)
g_vlinedraw(frm,srow,scol,erow,ecol,height,width,val)

gmrlib/vstring.c

g_vcstrdraw(frm,str,row,col,height,width,dir,rv,gv,bv)
g_vstrdraw(frm,str,row,col,g_height,g_width,dir,val)

gmrlib/zoom.c

g_getzoom(frm,factor,row,col,crosshair,blinking)
g_pan(frm,srow,scol,erow,ecol,trkdev,zfactor,crosshair,blinking)
g_setzoom(frm,factor,row,col,crosshair,blinking)
g_wrtzoom(factor,crsptr)
g_zoomoff()
g_zpan(currow,curcol,fflag,offlag)

piciolib/image.c

g_rdcimg(rimgd,gimgd,bimgd,frm,isrow,ierow,iscol,iecol,dsrow,dscol)
g_rdimg(imgd,frm,isrow,ierow,iscol,iecol,dsrow,dscol)
g_wrtcimg(rimgd,gimgd,bimgd,frm,dsrow,derow,dscol,decol,isrow,iscol)
g_wrtimg(imgd,frm,dsrow,derow,dscol,decol,isrow,iscol)

DESCRIPTION

Gmrlib is an adaptation of the CMU Grinnell image display system. The Testbed graphics system is currently built on top of it.

Gmrlib is based on a "frame" structure that is different from the Testbed FRM structure documented in gmrfrmlib(3).

The gmrlib package can be compiled for differing Grinnell configurations. The necessary customization for the IU Testbed Grinnell is in file gmrcnf.h. This is commonly copied to a temporary local file, libconfig.h, during compilation of display routines.

Grinnell I/O Buffering

The Grinnell Frame Buffer is controlled from software by reading and writing instructions and data to the device. Each routine in the Grinnell Library buffers its output so that each call to a library function will normally generate only one write instruction to the Grinnell.

However, it is often useful to be able to buffer the output to a greater extent. For example, a program may call setpixel in a loop several thousand times. Ordinarily, each call would generate a write instruction. Since a write is a system call, this can become quite slow.

In addition to the normal output mode, it is also possible to cause the software to buffer the output with the standard UNIX stream i/o routines. This may be done by calling g_setbuf with an argument of 1. Buffering is turned off by calling g_setbuf with 0 -- this automatically flushes the output buffer.

A factor of five speed increase is not uncommon for many consecutive calls to g_drawline (for example) when buffering. Note, however, that in most cases it is actually slower to use buffering because this involves copying the buffer of each routine into a global buffer. Buffering is efficient only in those cases in which fairly simple routines (such as setpixel and drawline) are called many times.

Changes to the CMU Package

See the CMU "Grinnell Display Software Support" document for further details. The described Grinnell configurations differ slightly from the Testbed configuration, and the compilation and loading instructions should be revised for our system. The following changes have also been made.

The g_setcur() routine has been changed on the testbed. The "mode" parameter has been replaced by two booleans, "visible" and "blinking". The "curstat" argument to g_getcur() has similarly been replaced. Routines g_curon() and g_curoff() have been eliminated. [They are easily replaced by calls to g_setcur() and g_getcur(), but eliminating them may have been a mistake. It interferes with running CMU code "native".]

The "zcursor" arguments to g_setzoom() and g_getzoom() have been replaced by three booleans: "crosshair", "blinking", and "zoomed". The g_zon() and g_zoff() calls have been eliminated.

The crsarg() routine has been added. It is a parsing routine that obtains a PNT structure (see pntlib(3)) from the program command line or from user interaction with the keyboard cursor controller. See arglib(3) for details.

FILES

/iu/tb/include/gmrcnf.h
/iu/tb/include/grinnell.h
/iu/tb/lib/imagelib/gmrlib/*
/iu/tb/lib/imagelib.a

SEE ALSO

dsplib(3), gmrfrmlib(3), gkeycur(3), gmrvdig(3)

DIAGNOSTICS

Each routine has its own messages and error codes. There are both compile-time and run-time debugging flags in some of the routines.

BUGS

The g_gmroff() command currently does not have an option to blank the overlays and cursors. Only the gray-scale image is blanked.

The piciolib/image.c routines access both images and the display device. All other routines in gmrlib are image-independent graphics routines.

Reading from the Grinnell in buffered mode is not feasible. If a read is attempted while buffering is turned on, the buffer will automatically be flushed.

The following changes from the CMU documentation should be noted.

> The ginit() command needs a monitor number as an argument. This should normally be 0 (also known as TB_MONITOR and G_TB_MONITOR).

The g_vline() and g_vcline() routines in the documentation are actually g_vlinedraw() and g_vclinedraw().

The G_ZP1 to G_ZP8 macros should be G_ZF forms instead.

G_intest(0) must be run before the other g_intest() calls in order to set up the overlays properly.

The programming example in Appendix II omits the argument to g_init() and the trailing ",0,0" arguments to g_rdimg(). Program comments assume an upper left origin, whereas the Testbed uses a lower left origin.

See the source directory and code headers for additional suggestions.

HISTORY

08-Dec-83 Laws at SRI-IU
Added the crsarg() routine.

08-Feb-83 Laws at SRI-IU
Created this **man** page.

04-Aug-80 Jerry Denlinger (jld) at Carnegie-Mellon University
Created g_setbuf().

NAME

gmrvdig – Grinnell video digitizer routines

SYNOPSIS

```
#include "ivdc.h"

g_vdon()
g_vdoff()

g_vdig(frame,shift,cont_mode,threshold,function,sumshf)
   int frame,shift,cont_mode,threshold,function,sumshf;

int g_vdsetup(frame,threshold,prompt)
g_vdsnap(frame,threshold)
   int frame,threshold;
   char *prompt;
```

DESCRIPTION

Low-Level Digitizer Routines

G_vdon(), g_vdoff(), and v_dig() are the low-level routines to control the Grinnell video digitizer. They should not be used in random programs, but only by subroutines written to do specific digitizing tasks.

G_vdon turns on the digitizer. That means that subsequent commands sent to the Grinnell will go to the digitizer. *G_vdig* sends commands to the digitizer causing it to perform the various things it can do. *G_vdoff* turns off the digitizer. Thus a program should call *g_vdon*, make one or more calls to *g_vdig*, and call *g_vdoff*, without issuing any other commands to the Grinnell in the meantime.

It is because of these constraints that it is strongly recommended that people not use these routines directly in programs. They should be used in procedures that turn on the digitizer, issue several commands, and turn it off, with no other digitizer commands issued in the meantime.

Frame is a frame descriptor returned by *g_allocfrm* or *g_splitframe*. The digitizer output will go into this frame. **Shift** is the number of bits (usually zero) that the digitizer output will be shifted down within the frame. When **shift** is zero, the most significant bit of the digitizer output is aligned with the most significant bit of the frame. The usual use of **shift** is to shift the output down in 7- or 8-bit frames or to shift the single-bit output of the digitizer when in threshold mode to other than the most significant bit of multiple-bit frames.

When **cont_mode** is zero, the digitizer will digitize one frame only; when it is nonzero, the digitizer will digitize continuously until another command is given to it. When **threshold** is less than zero, a gray-scale image is produced. If it is nonnegative, the image is thresholded at that value. Since the digitizer works to 6 bits of accuracy, it does not make sense to give a threshold greater than 63.

Function specifies what the digitizer will do with its output. Its values are defined in *g_vdc.h:*

G_VDREP

Replace the data in the frame with the newly digitized image.

G_VDSUM
: Ddd the digitizer output, pixel by pixel, to the data in the frame.

G_VDDIFF
: Subtract the digitizer output from the data in the frame.

G_VDweird
: Use the "weird" mode – see the Grinnell GMR 270 User's Manual.

Sumshf is a parameter used by the digitizer only when it is in "weird" mode. Once again, see the User's Manual.

High-Level Digitizer Routines

G_vdsetup() and g_vdsnap() allow the user to use the Grinnell video digitizer to digitize images and enable him to access most of the capabilities of the digitizer.

G_vdsetup runs the digitizer in continuous mode; i.e., the signal that is input to the digitizer is continuously digitized and displayed on the Grinnell monitor. If **threshold** is nonnegative, the digitizer is run in threshold mode, with the value of **threshold** as the threshold. Since the digitizer is accurate to 6 bits, it makes no sense to give it a threshold greater than 63. If **threshold** is less than zero, the digitizer output is gray-scale, using six bits or the size of **frame,** whichever is less.

Then *g_vdsetup* prints **prompt** and waits for the user to type *<newline>*, *<escape>*, or control-d. If the user types *<newline>*, it returns the value **G_GMRNOERROR;** otherwise it returns the value **G_GMRERROR.**

In either case, it stops the digitizer before returning, leaving the last image digitized on the display and in **frame.**

G_vdsnap takes a "snapshot"; i.e., it digitizes one Grinnell frame, leaving it in **frame** in the Grinnell. If **frame** is of fewer than six bits, pixel values are shifted down. If it has seven or eight bits, two or four successive video frames are added together (or averaged) to gain the desired precision. Just as in *g_vdsetup*, if **threshold** is less than zero, gray scale is used; if **threshold** is greater than zero, it is used as the threshold value for binary digitization.

In most cases, a call to *g_vdsnap* will be followed by a call to *g_wrtimg* in order to write the digital image stored in the frame buffer to an image file on the VAX.

Each of these routines performs a **g_chgdisp (frame).** Also, if threshold mode is used on a frame of more than 1 bit, the most significant bit is set to zero or one as appropriate, and the rest are set to zero.

EXAMPLE 1

The following program fragment will average four successive frames:

```
g_vdon();
g_vdig(frame, 2, 0, -1, G_VDREP, 0);
g_vdig(frame, 2, 0, -1, G_VDSUM, 0);
g_vdig(frame, 2, 0, -1, G_VDSUM, 0);
g_vdig(frame, 2, 0, -1, G_VDSUM, 0);
```

```
g_vdoff();
```

Note that each value is divided by four (shifted down two bits) before being summed, which may lead to loss of accuracy. It will not if the frame is 8 bits or if it comes from the most significant bits of an 8-bit hardware channel. That can be guaranteed by this code:

```
if (bits < 8) {
  garbageframe = g_allocfrm (mode, 8);
  g_splitframe (garbageframe, 8 - bits, &frame);
  }
else frame = g_allocfrm (mode, 8);
```

Frame will be a frame of **bits** bits, taken from the most significant bits in the channel that was originally allocated to **garbageframe**. **Garbageframe** will be converted to a frame of **(8 - bits)** bits located in the remainder of the channel originally allocated to it.

EXAMPLE 2

This is a program fragment that will allow a user to aim the camera and then specify a portion of the image to be written to disk. It does not check for errors.

```
bits = getint ("How many bits per pixel?", 1, 8, 8);
if (bits < 8) {
  garbageframe = g_allocfrm (G_BW, 8);
  g_splitframe (garbageframe, 8 - bits, &frame);
  }
else frame = g_allocfrm (G_BW, 8);
g_vdsetup (frame, -1, "<return> when ready.");
frs = getint ("First row of box?", 0, 511, 0);
fcs = getint ("First column of box?", 0, 511, 0);
fre = getint ("Last row of box?", frs, 511, 511);
fce = getint ("Last column of box?", fcs, 511, 511);
image = imgcreat ("vdoutput.img", 0644, bits,
                  0, fre-frs+1, 0, fce-fcs+1);
g_vdsnap (frame, -1);
g_wrtimg (image, frame, frs, fre, fcs, fce, 0, 0);
```

FILES

/iu/tb/lib/imagelib/gmrlib/include/ivdc.h
/iu/tb/lib/imagelib/gmrlib/gmrlib/vd*.c
/iu/tb/lib/imagelib.a

SEE ALSO

gmrfrmlib(3), *gmrlib(3)*,
IUS document CMU002, "Grinnell System Software Support," and the Grinnell GMR 270 User's Manual.

DIAGNOSTICS

G_vdsetup returns G_GMRERROR if the user aborted (with <esc> or EOF); it returns G_GMRNOERROR otherwise. *G_vdsnap* has no diagnostics.

BUGS

The digitizer will write on all 8 bits of whatever (hardware) channel its output is directed to, regardless of the number of bits allocated to the destination frame. Thus if arbitrary manipulation of frames is to occur, an eight-bit frame must be allocated, and the desired size frame must be split off of it with *g_splitframe*. The remaining "garbage" frame will be written on by the digitizer, so it should not be freed until the program is finished using the digitizer. See the latter part of the EXAMPLE section.

There is a bogus interaction between the digitizer and the overlay planes; once an overlay has been turned on, the digitizer will not work until the Grinnell is re-initialized.

HISTORY

12-Feb-83 Laws at SRI-IU
Combined previous **man** pages.

30-Sep-80 Steve Clark (sjc) at Carnegie-Mellon University
Updated and made ready for installation.

21-Aug-80 Steve Clark (sjc) at Carnegie-Mellon University
Created.

NAME

hdrlib – routines for manipulating picture headers

SYNOPSIS

#include "hdrlib.h"

Header descriptor (public fields only):

```
typedef HDR * struct {
    HEADERTYPE headertype;     Picture header format code.
    unsigned dataoffset;       Header length (first data byte offset).
    SCANTYPE scantype;         Scan direction code.
    BANDTYPE bandtype;         Pixel band format code.
    unsigned imagebands;       Bands per pixel.                 ( >= 0)
    unsigned bandbits;         Bits per band.                   ( >= 0)
    unsigned pixelbits;        Bits per pixel.                  ( >= 0)
    unsigned linepixels;       Pixels per scan line.            ( >= 0)
    unsigned imagelines;       Number of image scan lines.      ( >= 0)
    unsigned blockcols;        Bytes per block row.             ( >= 0)
    unsigned blockrows;        Rows (scan lines) per block.     ( >= 0)
};
```

```
HDR closehdr(hdr)
  HDR hdr;

HDR copyhdr(inhdr)
  HDR inhdr;

HDR newhdr([ncols,nrows,pixelbits,...]);
HDR newhdr([linepixels,imagelines,pixelbits,
     headertype,scantype,blockcols,blockrows,
     bandtype,imagebands,bandbits,dataoffset])
  HEADERTYPE headertype;
  SCANTYPE scantype;
  BANDTYPE bandtype;

HDR openhdr(filename,[&fullname])
  string filename;

printhdr(hdr)
  HDR hdr;

boolean validhdr(hdr)
  HDR hdr;

writehdr(fd,hdr)
  HDR hdr;
```

DESCRIPTION

An HDR is the parsed representation of a picture file header. Public fields are listed above; they should usually be treated as read-only. References to the fields should have the form "hdr->ncols", where hdr is declared to be of type HDR.

The hdrlib routines are primarily for the use of piclib utilities and should seldom be seen by the user. They may be useful, however, for determining the structure of a picture file without the overhead of opening the file for picture access.

Currently supported format codes are listed below. The defaults for normal Testbed use are TB01HDR, LRBTSCAN, and UNSIGNED pixels. For further information see imgformat(5) and hdrformat(5).

Picture header format codes:

```
typedef enum {
   NOHDR,               Raw data with no header.
   CMUHDR,              CMU format.
   CVLHDR,              Maryland CVL format.
   RVHDR,               Rochester RV format.
   SRIHDR,              SRI (Quam KL or VAX) format.
   TB01HDR              SRI TB01 testbed format.
} HEADERTYPE;
```

Scan direction codes:

```
typedef enum {
   NOSCAN,              Undefined or unknown pattern.
   LRBTSCAN,            Pixels left to right, bottom to top.
   LRTBSCAN,            Pixels left to right, top to bottom.
   RLBTSCAN,            Pixels right to left, bottom to top.
   RLTBSCAN             Pixels right to left, top to bottom.
} SCANTYPE;
```

Data type codes:

```
typedef enum {
   UNSIGNED             Unsigned integer.
} BANDTYPE;
```

Hdrlib subroutines:

closehdr(hdr) -> HDR
: If the HDR is multiply linked, simply decrement the link count; otherwise free the HDR structure. There is no effect on a picture file. The return value is generally NULL, but may be the original HDR value if an error occurred.

copyhdr(inhdr) -> HDR
: Copy all data associated with inhdr to a new HDR. The two HDRs will then be entirely separate.

newhdr(linepixels,imagelines,pixelbits,headertype,scantype,
 blockcols,blockrows,bandtype,imagebands,bandbits,dataoffset) -> HDR

Create a new HDR with the specified format. Zero or more arguments may be given, with the defaults being (512, 512, 8, TB01HDR, LRBTSCAN, 32, 32, UNSIGNED, 1, 8, 1024); thus, newhdr(128,128,8) might be a typical call. Linepixels and imagelines are equivalent to the number of picture columns and rows for horizontal scans (e.g., LRBTSCAN), the reverse for vertical scans.

Newhdr() may also be used to query the user for picture file parameters. If any argument is set to NOTINT, the user will be asked for the value. If a negative argument is given, it will be used as the default in asking for a value.

openhdr(filename,[&fullname]) -> HDR

Open the specified file and parse the header. This does not open the file for picture access. The PICPATH specified in your shell environment (default ":/iu/tb/pic:/aux/tbpic") is searched and, if &fullname is specified, the resulting full pathname is returned.

printhdr(hdr)

Print partial contents of an HDR structure. This is primarily used for debugging, although it may be of use in verifying to the user the success of opening a picture file.

validhdr(hdr) -> boolean

Check for a non-null HDR with a valid structure identification code.

writehdr(fd,hdr)

Write a picture file header to the open file fd.

FILES

/iu/tb/include/hdrlib.h
/iu/tb/lib/imagelib/hdrlib/*
/iu/tb/lib/imagelib.a
/iu/tb/lib/sublib.a
The math library (-lm) is also required.

SEE ALSO

blklib(3), doclib(3), imglib(3), piclib(3), imgformat(5), hdrformat(5)

DIAGNOSTICS

The hdrlib routines use the printerr error reporting system. An error will typically cause a message to be printed on the error stream and the routine will return a null HDR. If given a null HDR as an argument, the opening routines will generally ask for the needed information.

BUGS

Newhdr() allows a hdrlen of 0 to be specified for a TB01 header.

HEADERTYPEs other than TB01HDR are implemented for input but not for output. See shapeup(1) for format conversions. SCANTYPEs implying vertical scan are not currently working. BANDCODES other than UNTYPED are reserved but not yet implemented.

See the source directory for additional comments.

HISTORY

15-Nov-82 Laws at SRI-IU
Created this document.

NAME

icp – Interactive Command Processor

SYNOPSIS

#include "icp.h"

ICP();

DESCRIPTION

This document describes the Interactive Command Parser (hereafter known as "ICP"). ICP is a collection of modules (written in C) which gives a programmer the ability to access program variables and invoke functions interactively from a terminal. It is intended to be both a low-level checkout tool as well as a menu-driven command processor. Some of the important features are as follows:

- ICP provides a very C-like syntax. In fact, it behaves very much like a C interpreter (lacking control statements).
- ICP maintains a strong internal sense of variable types and can cast expressions into any of the primitive data types supported by C (e.g., "char", "int", "float").
- ICP contains a general arithmetic expression evaluator that closely follows C's order of evaluation. Parentheses may be used to alter the order of evaluation.
- Variables can be examined or changed interactively.
- Functions may be invoked interactively.
- Parameters may be passed to functions interactively.
- Function parameters may assume default values.
- Functions may be invoked automatically upon the alteration of pre-defined variables.
- Various menus of functions and variables can be "attached to" or "detached from" the current working menu.
- Temporary "scratch-pad" variables can be created interactively to contain the results of expressions.

Each of these features will be discussed in more detail below. An actual session with ICP has been recorded and stored in the file /iu/tb/doc/icp/tutorial.txt and may be referred to for additional information.

Talking to ICP at Runtime

At runtime, ICP will print a prompt and wait for the user to type a command. This command can take any of the following forms:

# xxxxxx
  [anything after "#" becomes a comment]

! xxxxxx
  [sends "xxxxxx" to shell as a command]

< filename
  [accepts ICP commands from "filename"]

> filename

	[sends user input to filename while executing]
?	[prints all known variables and functions]
xxx?	[prints all known variables and functions beginning with "xxx"]
exp	[evaluates "exp" as an expression (see below)].

Obviously, the inherent command set is quite concise and easy to remember. The power of ICP lies in its ability to evaluate expressions.

Evaluting Expressions at Runtime

As described above, ICP's main power comes from the types of expressions it can evaluate. To discuss the types of expressions that are acceptable, the following terms must be defined:

kvar	Any interactive variable in ICP's current menu.
uvar	Any temporary scratch-pad variable created at runtime (see below).
var	A kvar or a uvar.
fun	Any interactive function in ICP's current menu.
name	Any legal name for a variable; it must begin with an alphabetic character and be no more than 32 alphanumeric characters in length.
type	Any of the C primitive data types: char, int, short, long, unsigned, float, double, and void. Also supported are boolean (int), notype (void), and struct (of indeterminate type).
decl	{type} followed by any number of occurrances (up to six) of the character '*'. This defines a primitive data type or a pointer to a primitive data type (or up to six levels of pointer). Currently, the primitive struct must be accompanied by at least one '*'.

The final result of any expression evaluation will be expressed in terms of both the value of the expression and its data type. Using these definitions, acceptable expressions take the following forms:

var	On the right-hand side of a "=" sign, evaluates to the contents of that variable. On the left-hand side, it evaluates to the variable's address
fun	Invokes the given functions and evaluates to the value returned by that function
fun(e1,e2,...en)	Evaluates the given expressions (e1 - en) and invokes the given function with these expressions as arguments
&var	Evaluates to the address of that variable
&fun	Evaluates to the address of that function
dddd	A string of decimal digits evaluates to a long integer representing that decimal number
Oddd	A string of octal digits preceded by a "0" evaluates to a long integer representing that octal number
Oxddd	A string of hexadecimal digits preceded by an "0x" or "0X" evaluates to a long integer representing that hexadecimal number

ddd.ddd{E+-dd}
: A string of decimal digits containing a decimal point and optional exponent is evaluated to a double precision floating point number

'd'
: A single character enclosed in quotation marks evaluates to the numeric value of that ASCII character (C escape characters are acceptable. These take the form:

    ' ddd'

  where the '' character allows the following characters to be interpreted as an octal number giving the value for that character. Special escape codes such as in the C manual).

"xxxxxx"
: Any string of characters surrounded by quotation marks evaluates to the address of the first character in that string. A null (0) character is automatically appended to the string

boolean keywords
: "Yes," "true," and "on" evaluate to 1, "no," "false," and "off" evaluate to 0

pointer keyword
: The keyword NULL evaluates to 0

-e
: Evaluates the given expression and negates it

e1 = e2
: Evaluates "e2" and stores it in the location specified by "e1". The results of the expression are "e2"

*e
: On the right-hand side of an "=" sign, this evaluates to the contents of the datum addressed by the expression "e". On the left-hand side of the "=" sign, it evaluates to the address of that datum.

e1[e2]
: Logically equivalent to *(e1 + e2). The expression "e2" is multiplied by the size of the data type pointed to by "e1" (like C)

e1 {+-/*} e2
: Performs the specified binary operation on the two expressions and returns the result. If either expression is type "pointer," the other is multiplied by the size of the type of datum to which it points (and the result is type "pointer"). If neither is a pointer, both are converted to type "double" and the result is type "double"

(decl) e
: Evaluates the given expression and casts it to the type specified by "decl"

decl name
: Creates a temporary variable (uvar) of the specified type with the specified name. The result of the expression is the address of the newly created variable, allowing its appearance with other expressions

Warnings About Binary Operators

The binary operators ("*" for multiplication, "/" for division, "+" for addition, and "-" for subtraction) obey the basic rules of arithmetic in determining the sequence of evaluation. Parentheses can be used to reorder this sequence. Hence, the expression

```
10 + X * 13 / 12 - 11
```

is logically equivalent to

```
(10 +  (  (X * 13) / 12)   ) - 11.
```

As mentioned above, binary operators can produce two possible results, either a "pointer" expression or a "double" expression. If either of the operands is of type "pointer," the other is multiplied by the size of the data element to which the pointer points prior to the operation. The results of the operation are a pointer to that same data type. If neither is a pointer, both operands are converted to double-precision, floating-point numbers and the operation is performed. The result is an expression of type "double."

Because of assumptions made by the parsing mechanism, it is necessary to separate the binary operator from each of the operands by some white space. For example,

```
x + 20 / 30
```

is proper, whereas

```
x+20/30
```

will result in an error. In actual practice, only the "+" and "-" operators are subject to this limitation since they may appear within floating-point literals (e.g., 1.0E-10).

Warnings About String Arguments

A string expression evaluates to the address of the first character in that string. In practice, when ICP sees a string, it automatically appends a null (/000) onto the end of it and stores it in a a block of memory allocated (via malloc()). This memory remains allocated for the duration of the ICP session. It is never freed. Although this guarantees that any string will always remain accessible, it is somewhat wasteful of space. On a virtual address machine, this is considered acceptable.

Hence the expression

```
x = "hello"
```

can guarantee that the string "hello" will always be available and that "x" will contain its address.

Type Casting

The ability of ICP to cast an expression from one data type to another allows the user some flexibility in doing type-conversion at runtime. This type-conversion is always performed automatically to perform binary operations, to store expression results in interactive variables, or to pass parameters to functions. In general, any type conversion allowed by the C compiler is allowed. A warning message will print if the conversion seems a bit odd (converting a "char" to a "pointer," for example). Some conversions are simply not allowed (e.g., converting type "pointer" to "double"), and an error message will print that information.

Creating Temporary Variables (UVARS)

One of the most useful features of ICP is the ability to create temporary variables at runtime. These variables are called UVARS. Currently, 100 UVARS are allowed to exist simultaneously. The syntax for creating them is exactly like that used when programming in C. The following examples illustrate this:

char foo
    Creates a "char" UVAR named "foo"

int *Pointer_To_Int
    Creates a "pointer to int" named "Pointer_To_Int"

foo   Creates an "int" UVAR named "foo".

Once a UVAR has been created, it can be referred to in an expression just as any interactive variable can. When the user types "?" to get a list of available variables, UVARS are highlighted by printing a ':' after their name.

The primary difference between interactive variables and UVARS is seen when a value is assigned to them. Storing a value in an interactive variable will cause an automatic type-conversion so that the value will be of the proper type. Storing a value in a UVAR will cause no such type conversion. Instead, the UVAR will automatically alter its type to accommodate the unchanged value. The following examples illustrate this.

int foo
    Creates a UVAR name "foo" of type "int"

foo = 3.1415
    Changes "foo" to be type "double" and stores 3.1415 into it

foo = "hello"
    Changes "foo" to be type "pointer to char" and stores the address of the string into it

Once created, a UVAR cannot be deleted. However, to circumvent the problem of creating a UVAR with the same name as an interactive variable (which could only be done by creating the UVAR first and then attaching a menu containing an interactive variable of that name), an interactive variable with the same name will always take precedence over a UVAR in an expression.

UVAR's can be renamed by the "Rename-UVAR" command.

Note: the ability to create a UVAR simply by typing a string that ICP cannot otherwise recognize is a recent addition. It makes life a little easier, but also makes it extremely easy to create a UVAR inadvertently (e.g., by misspelling some other variable). The "Rename-UVAR" command was added to the standard menu to allow the user to change such a UVAR to a better name.

Passing Parameters to Functions

When an interactive function is defined (as described below under IFUN), an optional parameter list can be provided. This parameter list defines the type, name, and optional default value for each parameter for that function.

Functions can be invoked at runtime in either of two fashions--with arguments or without arguments. If arguments are specified, they will be matched against each of the parameters defined for the function. They will be cast into the appropriate type and passed to that function. Any missing arguments (detected by an end of line, a closing paren, or a bare comma) will assume the default value defined for that parameter. If no default value was defined, the operator will be asked to supply a value.

If a function is invoked with no arguments, ICP will supply the default values or query the operator for all defined parameters.

Parentheses may surround the parameters (or they may be omitted). Commas may separate the parameters (or they may be omitted). Note that parentheses should be used to indicate the end of the parameter list if the function call is nested inside a more complex expression.

Hence, the following are all examples of valid function calls:

```
fun            fun()          fun(
fun 1 2 3      fun (1 2 3)    fun (1 2 3
fun (1,2,3)    fun (,2,,)     fun 1 2 3)
```

How to Create Interactive Variables and Functions

The header file /iu/tb/include/icp.h defines four macros that are expected to be used by the programmer to define interactive variables and functions. They are IVAR, IFUN, IVARFUN, and ICMD. Each will be discussed separately.

The way in which these macros must be built into a menu will be described below as well.

The IVAR macro

The IVAR macro is used to define interactive variables. It takes the form:

```
IVAR ("decl name", addr)
```

"Decl" is a string containing a typical C-type declaration. "Name" is the alphanumeric name to be used to refer to this variable at runtime. "Addr" is the actual address of that variable. Hence, legal uses are:

```
double dd = {5.7};
char *cp = {"hello"};
...
```

```
IVAR("double A_Name_For_DD", dd),
ivar("char *A_Name_For_CP", cp),
```

The IFUN macro

The IFUN macro defines interactive functions. It takes the form:

```
IFUN("decl name", "arg1, arg2, arg3, ..., argN", addr)
```

As in IVAR, the "decl name" string supplies type information and a name for this function. The type information describes the type of data this function will return (which may be "void").

Following the "decl name" entry is a string describing each of the arguments as follows:

```
argN  :=      decl name {= e}.
```

That is, each argument also has a "decl name" entry to assign type information and a name to it. Optionally, the "decl name" entry may be followed by an "=" sign. If so, the "e" that follows will be assumed to be the default value for this argument. It is important to point out that this default value can be any expression the user might conceivably type at runtime, anything from a literal to the name of an interactive variable.

Arguments must be separated by commas. If a function has no arguments, a null string ("") or a NULL value (0) may be used in place of the argument list.

The "addr" entry defines the actual address of the given function.

The following are legal function definitions:

int foo() {return (0);} char *fooa(a,b,c) char *a; int b; double c; {return ("hello");}

```
IFUN("int Name_For_Foo", "", foo),
IFUN("int Another_Name_For_Foo", NULL, foo),
IFUN("char *Name_For_FooA",
    "char *arg1, int arg2, double arg3", fooa),
IFUN("char *Another_Name_For_FooA",
    "char *arg1 = "hi", int arg2 = 11, double arg3", fooa),
```

The IVARFUN macro

The IVARFUN macro is a derivative of the IVAR macro of the form:

```
IVARFUN("decl name", addr, funaddr).
```

The first two macro arguments are identical to IVARs. The third argument is the address of a function to be invoked whenever the indicated variable is modified. Legal forms are:

```
        double dd = {5.5};
        foo() {if (dd > 6.6) dd = 5.5;}

        IVARFUN("double Name_For_DD", dd, foo),
```

The function will not be called unless the variable is actually modified. If the variable is modified (even to contain the same value), the function will be called. If ICP can find the given function address anywhere in its connected menus, it will invoke that function as if the user had invoked it with an empty argument list. This means that if the function address can be found in the menu, parameters may be passed to that function. Default values will be used if they are defined; otherwise the user will be asked to supply them. If ICP cannot find the function address in the current menu, it will invoke that function address with no arguments.

The ICMD macro

The ICMD macro allows a function to be given an unparsed list of arguments. It takes the form

```
        ICMD("decl name", "string", addr).
```

In appearance, it is exactly like IFUN, except for the second macro argument, which is a string that will be printed only for user information. It does not define the parameters in any way. When an ICMD function is invoked, the entire remainder of the command line is separated into an argv[] style list structure (a common C structure). The function is invoked with two arguments, the number of entries in the argv[] list, and a pointer to the argv[] list. An example:

```
    int foocmd (argc, argv)
     int argc;
     list argv;
    {
     int k;
     for (k = 0; k < argc; ++k)
        printf("0rgv[%d] = %s", k, argv[k]);
     return (argc);
    }

    ...

    ICMD("int foocmd", "Anything you want printed on the menu", foocmd),
```

Building Interactive Variables and Functions In a Menu

Obviously, these macros cannot appear just anywhere in the program. Instead, they must be built into an array in the following manner:

```
struct ICENTRY mymenu [] = {
 ivar(...),
 ifun(...),
 ivarfun(...),
 icmd(...),
 iend
};
```

The "ICENTRY" structure is defined in icp.h and is, in fact, the structure to which all the macros refer. The "iend" macro, a special macro to identify the end of a menu, MUST be present. It takes no arguments.

As many menus as are desired may be built in this fashion. ICP will handle an unlimited number of separate menus.

The order of the entries in this array is not important. Typically, the programmer will combine IVAR's and IFUN's dealing with related modules of code into one menu, but is free to organize the menu according to individual preference. The number of entries in any menu is unlimited.

Attaching These Menus to ICP

To attach these menus to ICP, it is necessary for the program to call an ICP function. For flexibility, three functions are available. Their names and arguments are:

```
AddTab(menu);
RmvTab(menu);
AvailTab(menu, "menuname");
```

The "AddTab" function will simply attach the given menu (where "menu" is the address of the array of macros). Such an attachment can be considered permanent (unless AvailTab() is also invoked). Whenever the user types "?," all the interactive variables and functions defined in this menu will be displayed. Any number of menus may be attached in this fashion.

The "RmvTab" function is the complement of "AddTab." It will remove a menu from the working menu.

The "AvailTab" function simply associates an ASCII name with a given menu. It does not attach that menu. At runtime, the user can interactively attach and/or detach any menus that have been made available in this fashion. Currently, only about 100 menus may be given names through "AvailTab." Note that this is the preferred method of making menus available to the user, who can attach needed menus or detach unneeded ones to create a personal working menu.

Invoking ICP

After attaching any menus desired, the user program must invoke the function

ICP() using the form:

```
        result = ICP();
```

ICP is essentially a loop. It may be termintated by user command, and it will return a value (specified by the user, or zero by default).

The Initial Menu

When ICP has been invoked, it will print its prompt "->" and accept commands. Six commands are initially attached to the working menu:

```
        Attach-menu
        Detach-menu
        Menu
        Exit
        Rename-UVAR
        Verbosity.
```

The "Attach-menu" command (which was actually defined using the "ICMD" macro) will accept any number of menu names to attach. These names must be the same names given to AvailTab(). It will invoke AddTab() for each name it can recognize. This is the manner in which the user makes menus available at runtime. Invoking "Attach-menu" with no arguments will cause the list of available menus to be printed.

The "Detach-menu" command is the complement of "Attach-menu." It will accept any number of menu names to remove from the working menu. It will invoke RmvTab() for each menu name it recognizes. These menus may later be reattached through "Attach-menu."

The "Exit" command will cause ICP to return to the calling program. It will accept one integer argument expressing the value to be returned. The default value is zero.

The "Menu" command is like "Attach-menu" and "Detach-menu" in that it will accept any number of menu names. Rather than attaching or deetaching those menus, however, it will merely list all the elements in that menu (whether or not that menu is attached). It functions exactly as if the menu has been attached and the user had typed '?'. It allows the user to browse through menus without first attaching them.

The "Rename-UVAR" command accepts any number of arguments in the form "Rename-UVAR a b c d ... y z". It will interpret the first argument of any pair of arguments as an old UVAR name and the second argument as the new UVAR name. In the example shown, it would rename "a" to "b", "c" to "d", and "y" to "z".

The "Verbosity" variable allows the user to select the amount of interactive output generated by ICP. The following values have meaning:

0: inhibits all normal interactive output (not error messages)
1: allows only a small amount of interactive output
2: allows all interactive output.

Predefined Libraries

Several libraries are made available to ICP (via AvailTab()) when it is invoked. Currently, they are:

IO_Library
Math_Library
String_Library
System_Library

These libraries were created by using the standard macros ("IVAR", "IFUN", etc.) to give the user access to many of the same functions that would be available when programming (e.g., "strcpy", "malloc", "fopen", etc.). They will grow as people want more entries, so I won't detail their contents here.

The user does not have to do anything special to get these libraries. They are defined in /iu/tb/lib/icplib/icplib/libic.c and attached to ICP automatically. They must be explicitly attached using the "Attach-menu" command at runtime, however.

FILES

/iu/tb/include/icp.h
/iu/tb/lib/sublib/icplib/*
/iu/tb/lib/sublib.a

The directory /iu/tb/lib/sublib/icpdemo contains a demonstration program that invokes ICP. We recommend that you look at file demo.c and see how all the items described above are actually used.

The file "demo" is a runnable version of this code and may be tried to get a feel for ICP.

The "makefile" links demo.c with ICP to make pdemo. It will provide an example of the linking requirements.

File icpdemo.txt records a session with the ICP demo.

SEE ALSO

dpy(1), imgsys(1), ci(3)

BUGS

There is no interactive help facility.

Negative reals don't work. "foo -1.0 2.0" ignores the second argument and prompts for it! [This may have been fixed.]

If you hit CR in response to an argument prompt, it aborts the entire command.

Expansion of abbreviated names currently takes precedence over creation of a uvar by assignment. Having a function named "image" prevents one from creating a uvar named "i" by typing "i = 10".

A uvar may be created before the value has been computed without error; this can produce a garbage uvar if you type the value incorrectly.

See the source directory for additional remarks.

HISTORY

07-Feb-83 Laws at SRI-IU
Created this file from one supplied by Ron Cain of SRI.

NAME

imagemac – CMU image handling macros

SYNOPSIS

#include "imagemac.h"

PROGRESS(row,verbose)
int row,verbose

DESCRIPTION

Imagemac provides three services: (1) macros for more compact access to information in IMAGE structures; (2) flag bits for the *imgshift* procedure; and (3) progress marks.

The following are valid only for programs using CMU image access. Testbed PIC and IMG descriptors are entirely different.

Macro call	**Effect (less parentheses)**
rows(img)	img->IM_rows
cols(img)	img->IM_cols
rstart(img)	img->IM_rstart
cstart(img)	img->IM_cstart
pixbits(img)	img->IM_pixbits
rend(img)	img->IM_rend
cend(img)	img->IM_cend
IMfilename(img)	img->IM_filename

The following macros are defined for *imgshift* (see cmuimglib(3)), which requires two flag bits:

```
#define IM_TEMPORARY    0
#define IM_RELATIVE     0
#define IM_PERMANENT    1
#define IM_ABSOLUTE     2.
```

Imgshift(img,1,1,IM_PERMANENT|IM_ABSOLUTE), for example, sets the origin to (1,1) in the external file.

PROGRESS prints progress marks on the standard output. If *verbose* is true and *row* is a multiple of ten, then the hundreds' digit of *row* is printed on the standard output.

FILES

/iu/tb/include/imagemac.h

SEE ALSO

cmuimglib(3)

HISTORY

20-Apr-81 Steve Clark (sjc) at Carnegie-Mellon University
Removed IMdescription (obsolete and nonfunctional).

19-Mar-80 David Smith (drs) at Carnegie-Mellon University
Created.

NAME

imgfrmlib – image and picture display routines

SYNOPSIS

```
#include "imglib.h"
#include "frmlib.h"

IMGTYPE dftfrmimg(frm)
   FRM frm;

FRMTYPE dftimgfrm(img)
   IMG img;

boolean imgfrmok(img,frm)
   IMG img;
   FRM frm;

showimg([img,frm,frmwdw])
   IMG img;
   FRM frm;
   WDW frmwdw;

showpic([pic,frm,frmwdw])
   PIC pic;
   FRM frm;
   WDW frmwdw;

showpicwdw(pic,picwdw,frm,frmwdw)
   PIC pic;
   WDW picwdw;
   FRM frm;
   WDW frmwdw;
```

DESCRIPTION

These routines use both the image access code (through IMG or PIC structures) and the Testbed display frame (FRM) system.

Showimg() and showpic() insert data into a display frame, but do not necessarily display it. Use showfrm() (see frmlib(3)) to make the frame visible.

dftfrmimg(frm) -> IMGTYPE
: Return the default image type for a given frame. This is useful for constructing interactive queries. See imglib(3) for the defined IMGTYPE codes.

dftimgfrm(img) -> FRMTYPE
: Return the default frame type for a given image. This is useful for constructing interactive queries. See frmlib(3) for the defined FRMTYPE codes.

imgfrmok(img,frm) -> boolean
: Check compatibility between a given image and display frame.

showimg([img,frm,frmwdw])
: Display an image in a frame window. If img or frm is NULL or omitted, the user will be asked for the needed information. If frmwdw is NULL or omitted, the whole frame will be used.

  If the image window is too large for the frame window, only the lower left corner will be displayed. (A warning will be printed.) If the frame window is too small, it will be erased before inserting the picture.

showpic([pic,frm,frmwdw])
: Display a picture in a frame window in the same manner as showimg(). Since a PIC has no associated window, the lower left corner is always displayed.

showpicwdw(pic,picwdw,frm,frmwdw)
: Display a picture window in a frame window in the same manner as showimg().

EXAMPLE

For a simple example of showpic() usage, see /iu/tb/lib/imagelib/demo/showpic.c. For showimg(), see /iu/tb/src/show/show.c, a Testbed routine with full argument parsing and error checking.

FILES

/iu/tb/include/frmlib.h
/iu/tb/include/imglib.h
/iu/tb/lib/imagelib/imgfrmlib/*
/iu/tb/lib/imagelib.a
/iu/tb/lib/sublib.a

SEE ALSO

frmlib(3), imglib(3), piclib(3)

DIAGNOSTICS

These routines use the printerr error reporting system. An error will typically cause a message to be printed on the error stream and a printerr return code will be set.

BUGS

Stereo images are displayed as color images with a null third feature plane.

See the source directory for additional comments.

HISTORY

15-Nov-82 Laws at SRI-IU
: Created this file.

NAME

imglib – high-level picture and image access

SYNOPSIS

#include "imglib.h"

Image descriptor (public fields only):

```
typedef IMG * struct {
    IMGTYPE imgtype;          Image type code.
    int ncols;                Number of columns.  [Not affected by
    int nrows;                Number of rows.        windowing.]
    int pixelbits;            Pixel size after conversion.
    BANDTYPE pixeltype;       Data type code.
    IMGNME imgnme;            Output image name.
    PIC *pic;                 List of PICs.
    WDW wdw;                  WDW into the PICs.  [Always defined.]
};
```

IMG closeimg(img)
  IMG img;

IMG linkimg(inimg)
  IMG inimg;

IMG newimg([imgtype,hdr,filename0,...])
  IMGTYPE imgtype;
  HDR hdr0;
  string filename0,...;

IMG openimg([imgtype,pic0,...][,imgmode])
IMG openimg([imgtype,filename0,...][,imgmode])
  IMGTYPE imgtype;
  PIC pic0,...;
  string filename0,...;
  string imgmode;

printimg(img,[verbose])
  IMG img;
  boolean verbose;

IMG scrapimg(img)
  IMG img;

setimgwdw(img,wdw)
  IMG img;
  WDW wdw;

boolean validimg(img)
  IMG img;

DESCRIPTION

An IMG is the handle (or "capability") by which a multiband image is accessed. Public fields are listed above. They should usually be treated as read-only. References to the fields should have the form "img->ncols", where img is declared to be of type IMG. The associated PIC fields may be accessed by img->pic[0]->ncols and the current window dimensions by img->wdw->ncols.

The public information in the IMG is essentially copied from the constituent PICs. The imgnme field holds a composite of all the PIC filenames and a list of implied feature names (e.g., an image opened by openrgbimg() will have features "red", "green", and "blue" regardless of the true PIC filenames).

The image dimensions, ncols and nrows, represent the maximum allowable subscripting range, whereas the associated WDW delimits the current context. All image coordinates are zero-based, beginning with (0,0) in the lower left corner.

The IMGTYPE codes are listed below. For BANDTYPE codes see hdrlib(3) or the Testbed picture header documentation.

Image type codes:

```
typedef enum {
   NOIMG,              Undefined image type.
   BWIMG,              BW (monochrome) image.
   LRIMG,              Stereo (lft, rgt) pair.
   RGBIMG,             RGB color image.
   HSIIMG,             HSI color image.
   YIQIMG              YIQ color image.
} IMGTYPE;
```

Imglib subroutines:

closeimg(img) -> IMG
: If the IMG itself (as opposed to its constituent PICs) is multiply linked, simply decrement the link count. Otherwise close the picture files and free the IMG structure. Writable images that are not closed will usually be truncated. The return value is generally NULL, but may be the original IMG value if an error occurred.

linkimg(inimg) -> IMG
: Copy the input IMG and link to all the associated PICs. The two IMGs will then access the same data. The PICs know they have been linked, so that closing or "scrapping" one IMG will have no effect on the files used by the other.

newimg([imgtype,hdr,filename0,...]) -> IMG
: Create an image of the indicated type (see above). The hdr field permits image dimensions and data format to be specified. The number of filenames required depends on the type of image. Each will be created as

a PIC with read/write access. (This may overwrite on existing files.) Missing or NULL fields will be requested interactively.

openimg([imgtype,pic0,...][,imgmode]) -> IMG
openimg([imgtype,filename0,...][,imgmode]) -> IMG
Open an image of the indicated type (see above). The number of PICs or filenames required depends on the type of image. Valid PICs will simply be attached to the new IMG; anything else will be treated as a file name and will be opened with the specified access mode ("r", "w", or "rw"; default "r"). If NULL, the user will be asked for a file name. The PICPATH environment variable (default ":/iu/tb/pic:/aux/tbpic") is used as a search path if necessary.

printimg(img,[verbose])
Print partial contents of a IMG structure. This is primarily used for debugging. The verbose field may be specified to get a (nonverbose) description of each PIC in the image.

scrapimg(img) -> IMG
Close the image and delete the picture files (unless they are multiply linked).

setimgwdw(img,wdw)
Redefine the window associated with the image. The new window may be taken from another image or generated by a call to newwdw() (see wdwlib(3)). This routine copies the passed window instead of linking to it in order to avoid side effects. The window is currently restricted to lie within (0,img->ncols) and (0,img->nrows), which is the window defined by openimg().

validimg(img) -> boolean
Check for a non-null IMG with a valid structure identification code.

EXAMPLE

For a sophisticated example of imglib usage, see /iu/tb/src/show/show.c.

FILES

/iu/tb/include/imglib.h
/iu/tb/lib/imagelib/imglib/*
/iu/tb/lib/imagelib.a
/iu/tb/lib/sublib.a
The math library (-lm) is also required.

SEE ALSO

hdrlib(3), imgfrmlib(3), imgnmelib(3), piclib(3)

DIAGNOSTICS

These routines use the printerr error reporting system. An error will typically cause a message to be printed on the error stream and a printerr return code to be set. A NULL IMG is usually returned.

BUGS

The ncols and nrows fields are currently taken from the first PIC. An intersection of all the PICs would be preferable.

See the source directory for additional comments.

HISTORY

01-Feb-83 Laws at SRI-IU
Changed the arguments to newimg(), printimg().

26-Nov-82 Laws at SRI-IU
Changed windowimg() to setimgwdw().

15-Nov-82 Laws at SRI-IU
Created this file.

NAME

imgnmelib – routines for image name manipulation

SYNOPSIS

#include "imgnmelib.h"

Image name descriptor (public fields only):

```
typedef IMGNME * struct {
 string path;              Generic image directory path.
 string generic;           Generic image name.
 int resolution;           Resolution factor.
 list bandlist;            Feature (or data band) names.
 string extension;         File extension.
};
```

IMGNME closeimgnme(imgnme)
  IMGNME imgnme;

IMGNME copyimgnme(inimgnme,[path,generic,resolution,bandlist,extension])
  IMGNME inimgnme;
  string path,generic;
  list bandlist;
  string extension;

IMGNME imgnmearg(cmdarg,flagvct,dftstr,[askmsg])
  boolean flagvct[5];
  string dftstr,askmsg;

string imgnmestring(imgnme,[band])
  IMGNME imgnme;

IMGNME jointimgnme(imgnme0,imgnme1)
  IMGNME imgnme0,imgnme1;

IMGNME linkimgnme(inimgnme)
  IMGNME inimgnme;

IMGNME mergeimgnme(imgnme,dftimgnme)
  IMGNME imgnme,dftimgnme;

IMGNME newimgnme(path,generic,resolution,bandlist,extension)
  string path,generic;
  list bandlist;
  string extension;

IMGNME parseimgnme(namestr)
  string nmestr;

string picnmestring(imgnme,[path,generic,resolution,bandname,extension])
  IMGNME imgnme;
  string path,generic;

```
    string bandname,extension;
```

printimgnme(imgnme)
```
    IMGNME imgnme;
```

boolean validimgnme(imgnme,[flagvct,warnflag])
```
    IMGNME imgnme;
    boolean flagvct[5],warnflag;
```

DESCRIPTION

An IMGNME is a parsed representation of a CMU-style image name. Public fields are listed above. They should usually be treated as read-only. References to the fields should have the form "imgnme->path", where imgnme is declared to be of type IMGNME. The associated "feature" or "band name" fields may be accessed as imgnme->bandlist[0], etc.

The generic image itself is a directory containing various picture files, such as the red, green, and blue picture files of a color image or the several picture files of various resolutions in an image "pyramid." An IMGNME represents this as five fields:

path
: The directory path of the generic image, not including the image name itself.

generic
: The generic image name.

resolution
: The resolution of the particular image file. This field is NOTINT if it is null in the image name. A resolution of 1 is the original; resolution of n indicates that n*n pixels in the original map to one pixel in this image file.

bandlist
: A list of the features contained in this image file, such as red, blue, density, hue. The list is NULL if no features are specified.

extension
: The extension of the image file. The extension is to contain miscellaneous information; normally it should be img, dat, or null.

The information in the IMGNME is usually copied from a set of PIC names. Fields other than bandlist hold the consistent descriptors, or NULL if the PICs are inconsistent (e.g., have different generic names). NOTINT is used for inconsistent resolutions.

The bandlist field is a C-style list of feature names. For output image names these feature names are often assigned by the program (e.g., an image opened by openrgbimg() will have features "red," "green," and "blue" regardless of the original PIC filenames).

Imgnmelib subroutines:

closeimgnme(imgnme) -> IMGNME

If the IMGNME is multiply linked, simply decrement the link count; otherwise free the IMGNME structure. The return value is generally NULL, but may be the original IMGNME value if an error occurred.

copyimgnme(inimgnme,[path,generic,resolution,bandlist,extension]) -> IMGNME
: Copy all fields of the image name to a new IMGNME structure. The remaining arguments, if specified and not NULL or NOTINT, will override those in the original IMGNME.

imgnmearg(cmdarg,flagvct,dftstr,[askmsg]) -> IMGNME
: Interactively obtain a CMU-style image name. The flag values in flagvct indicate whether the corresponding IMGNME fields are required (TRUE), optional (NOTBOOL), or required to be absent (FALSE). Dftstr may be NULL if there is no default. Askmsg may be NULL, a string, or a LST of prompt strings (initial prompt, image name prompt, and additional help message). See arglib(3) for further details.

imgnmestring(imgnme,[band]) -> string
: Convert an IMGNME to a string. If "band" is specified, or if there is only one feature band, the string will be a complete file name (assuming that the required information is present in the IMGNME). Otherwise the feature component will be omitted and the string will be a generic image name.

jointimgnme(imgnme0,imgnme1) -> IMGNME
: Combine two IMGNMEs by taking the intersection of their path, generic, resolution, and extension fields (i.e., delete any conflicting values) and the union of their feature names. The feature names are simply concatenated, not merged.

linkimgnme(inimgnme) -> IMGNME
: Link to an existing IMGNME structure. Any change to one IMGNME will also affect the other, but they may be closed independently.

mergeimgnme(imgnme,dftimgnme) -> IMGNME
: Fill out missing fields in an IMGNME with corresponding fields from a default IMGNME.

newimgnme(path,generic,resolution,bandlist,extension) -> IMGNME
: Create an image name structure with the specified fields.

parseimgnme(namestr) -> IMGNME
: Parse a file name or generic image name to obtain the IMGNME fields.

picnmestring(imgnme,[path,generic,resolution,bandname,extension]) -> string
: Convert an IMGNME to a string. If "bandname" is specified, or if there is only one feature band, the string will be a complete file name (assuming that the required information is present in the IMGNME). Otherwise the feature component will be omitted. You may omit the overriding

arguments or set them to (,NULL,NULL,NOTINT,NULL,NULL).

printimgnme(imgnme)
: Print the contents of an IMGNME structure. This is primarily used for debugging. If multiple features are present, they will be listed after the generic name.

validimgnme(imgnme,[flagvct,warnflag]) -> boolean
: Check for a non-null IMGNME with a valid structure identification code. The flag values in flagvct indicate whether the corresponding IMGNME fields are required (TRUE), optional (NOTBOOL), or required to be absent (FALSE). Warnflag may bet set to print a specific warning about any field that violates these constraints.

EXAMPLE

For an example of IMGNME usage, see /iu/tb/src/convert/convert.c.

FILES

/iu/tb/include/imgnmelib.h
/iu/tb/lib/imagelib/imgnmelib/*
/iu/tb/lib/imagelib.a
/iu/tb/lib/sublib.a

SEE ALSO

arglib(3), imglib(3), cmunmelib(3), stringlib(3)

DIAGNOSTICS

These routines use the printerr error reporting system. An error will typically cause a message to be printed on the error stream and a printerr return code will be set. A NULL IMGNME is usually returned.

BUGS

Jointimgnme() should accept any number of arguments.

It is not known yet whether linking to an IMGNME is a useful operation. It is provided for consistency with the blklib package.

HISTORY

01-Feb-83 Laws at SRI-IU
Created this file.

NAME

keycur – Grinnell keyboard cursor driver

SYNOPSIS

g_keycur(curnum,lpr0,srow,scol,erow,ecol,argflg,usrfnc)
```
  int curnum;
  short unsigned lpr0;
  int srow,scol;
  int *erow,*ecol;
  int *argflg;
  boolean (*usrfnc)();
```

DESCRIPTION

Use the keyboard joystick emulator to read pixel values. Key "a" prints the image value at the current cursor position.

Keys in the upper keypad configuration jump the cursor to corresponding home positions. (Motion may be restricted by the size of the currently displayed image.)

```
^[  ^\  ^]
{   |   }
*   +   ^C
```

Incremental cursor motion is controlled with the lower keypad. An "=" returns the cursor to its initial position. The motion step size begins at 10, and can be reset to 1-, 10-, or 100-pixel steps using the "[", "\", and "]" keys.

```
  (    ^    )
  <    =    >
Clear  LF  Hold
```

You may also type "w" to read the current position, "e" to enter a new position, and "f" to read the current flag settings. Keys "a" and "b" are used to set flag values, "A" and "B" to turn them off. (The cursor routine uses flag "a" to request the current coordinates and image value. The flag is then reset.)

A carriage return or "z", "Z", or "Control-Z" returns the current cursor position and flag values to the calling routine. "q" returns a QUIT status code and restores the cursor and flag settings to the original entry values. "Control-Y" may be used to quit and terminate program execution.

Returns OK for a "finished" input, QUIT for a "quit" input. Flag should be initialized before entry. Lpr0 is the Grinnell quad cursor register 0 word; try 0x0001. Usrfnc() is a user-supplied routine that may examine the cursor state after each point is entered. Supply NULL or 0 if this is not needed. (See the source code for more details.)

FILES

/iu/tb/lib/imagelib/gmrlib/crslib/keycur.c
/iu/tb/lib/imagelib.a

SEE ALSO

ghough(1), dsplib(3), gmrfrmlib(3), gmrlib(3)

BUGS

All Grinnell library routines start with "**g_**"; thus the name of this manual page does not quite correspond to the name of the actual library routine.

The use of lpr0 to pass Grinnell register codes makes this routine very device-dependent.

The keypad arrangement is only optimal for a Datamedia 3025.

The g_keycur() routine does not have an argument to allow situation-dependent setting of the initial step size.

The use of the center key to return the cursor to its initial position also needs to be rethought. This is occasionally handy, but usually annoying (when hit by accident). Perhaps repeated center key strokes should toggle between positions.

HISTORY

07-Feb-83 Laws at SRI-IU

Created this documentation file for the modified Testbed keyboard cursor. The original code is from CMU.

NAME

listlib – routines for manipulating C lists

SYNOPSIS

```
#include "listlib.h"

list addlist(inlist,[element0,...])
   list inlist;
   string element0;

list appendlist(list0,list1)
   list list0,list1;

int bestmatch (stringarg,srchlist,quiet)
   string stringarg;
   list srchlist;
   boolean quiet;

list buildlist(helppath,template,length)
   string helppath,template;

list copylist(inlist)
   list inlist;

int expand(filespec,filelist,listsize)
   string filespec;
   list filelist;

freelist(inlist)
   list inlist;

int listlen(listarg)
   list listarg;

int listmatch(stringarg,srchlist,quiet)
   strint stringarg;
   list srchlist;
   boolean quiet;

list mergelist(list0,list1)
   list list0,list1;

list newlist(element0,...)
   string element0;

printlist(listarg)
   list listarg;

list striplist(filelist)
   list filelist;
```

DESCRIPTION

A list is a vector of pointers in the manner of argv, or equivalently a pointer to a vector of strings. The actual C definition is

```
typedef string *list;
```

By convention, the last valid string pointer must be followed by a NULL or 0. Preceding strings should not be NULL unless you keep track of the list length, as with a blklib LST header.

You may create and initialize a list using the syntax

```
string foo[] = {"fee","fie","foe","fum",0};
```

Unfortunately, the compiler will not let you say

```
list foo = { ... };
```

as you can with strings, but it is otherwise equivalent. The keyword "static" should be used unless the declaration is global and the variable is to be made available externally. Any external reference to this structure should specify a string[], but argument declarations may call it a list.

addlist(inlist,[element0,...]) -> list
: Return a new list that has the specified elements appended to those of inlist.

appendlist(list0,list1) -> list
: Return a new list made by appending the input lists.

bestmatch (stringarg,srchlist,quiet) -> int
: Find the element in srchlist that best matches stringarg. A match quality score is used for inexact matches. The quiet flag may be set to suppress interactive help on bad matches. Returns -2 for for an ambiguous match, -1 for no match, and a list index otherwise.

buildlist(helppath,template,length) -> list
: Build a list of filenames matching the template; the maximum length of the list is also specified. This is used by the CI driver package.

copylist(inlist) -> list
: Copy all fields of the list to a new list. (Only the pointers are copied, not the objects pointed at.)

expand(filespec,filelist,listsize) -> int
: Expand a file specification by resolving the characters '*', '?', and '[' (also ']') in the same manner as the shell. You provide "filelist", which is a C-style list, and you tell how big it is in listsize. Expand will compute the corresponding filenames, and will fill up the entries of filelist, putting pointers to dynamically created strings into the slots.

: The value returned by expand is the number of filenames found. If this value is -1, then some error occurred and not all of the files were discoverable. If the value is listsize+1, then too many names were found; you should try again with a bigger list vector.

freelist(inlist)
: Free all the elements of a list. Currently this does not free the list itself. Be sure the strings in the list were dynamically allocated before you call this.

listlen(listarg) -> int
: Return the length of a list excluding the final NULL.

listmatch(stringarg,srchlist,quiet) -> int
: Find the element in srchlist that is matched by stringarg. Matching is exact except that stringarg may be a leading substring. The quiet flag may be set to suppress interactive help on bad matches. Returns -2 for an ambiguous match, -1 for no match, and a list index otherwise.

mergelist(list0,list1) -> list
: Append two lists, eliminating duplications. The new list is not sorted.

newlist(element0,...) -> list
: Create a new list with the specified elements. Currently there is no way to specify the allocated length; it will be just large enough to hold the list elements and a final NULL.

printlist(listarg)
: Print the elements of a list in multicolumn format. It is assumed that the elements are strings, so be careful.

striplist(filelist) -> list
: Copy a list of filenames with the paths stripped off.

FILES

/iu/tb/include/listlib.h
/iu/tb/lib/sublib/listlib/*
/iu/tb/lib/sublib.a

SEE ALSO

lstlib(3), stringlib(3)

DIAGNOSTICS

Some of the listlib routines use the printerr error reporting system. An error will typically cause a message to be printed on the error stream and the routine will return a null list. Other routines (e.g., bestmatch() and listmatch()) use special return codes to flag errors.

REMARKS

For most purposes a list and a string[] are equivalent. (The compiler converts all array references to array >>address<< references, and all subscripts to address offsets.) The only difference between them is the compiler's willingness to initialize a string[] and to take the address of a list or return a list as a function value.

Suppose you have the following:

```
string listA[] = {"foo","bar","baz",0};
list   listB   = listA;
```

You may then print these stuctures identically:

```
printlist(listA);
printlist(listB);
```

To access the first string of listA you may use *listA or listA[0], and likewise for listB you may use *listB or listB[0]. You may take the address of any of these string elements.

If a subroutine required the >>address<< of a list, things would be more difficult. You cannot pass down &listA since the compiler considers this a redundancy: it gives you a warning message and &listA[0]. You can pass &listB, and it differs (as it should) from &listB[0]. Thus you may easily pass the address of a "list", but not of a string[].

BUGS

Beware of naming any variable "list", since the compiler will try to expand it as a type declaration. This applies to any routine which includes testbed.h directly or indirectly.

Bestmatch() and listmatch() allow no more than 500 strings in the srchlist.

Bestmatch() does not resolve multiple matches very well. It should give preference to leading substrings over interior substrings, and should choose the best match even when "quiet" is set.

Printlist has some hard-wired constants. It assumes that there are 80 columns on the page and that a leading tab is 7 or 8 spaces. It leaves 5 spaces more than the longest of all strings between columns, and likes to print at least 8 rows if there are just a few strings. At least the indentation should be user-controlled.

See the source directory for additional comments.

HISTORY

02-Feb-83 Laws at SRI-IU
Created this document.

21-Dec-81 Laws at SRI-IU
Changed names from stabsrch to bestmatch, stablk to listmatch, and prstab to printlist.

27-Jan-81 Steven Shafer (sas) at Carnegie-Mellon University
Added better handling of long string srchlists.

22-May-80 Steven Shafer (sas) at Carnegie-Mellon University
Added check for exact match if more than one string matches by listmatch().

16-Apr-80 Steven Shafer (sas) at Carnegie-Mellon University
Created printmatch and used it in listmatch().

15-Mar-80 Steven Shafer (sas) at Carnegie-Mellon University

Lstsrch now detects a unique match with the initial characters of an optlst entry, and returns the index without asking "Did you mean X?"

05-Dec-79 Steven Shafer (sas) at Carnegie-Mellon University
Created listmatch() and bestmatch() (under the names stablk() and stabsrch()) based on Dave McKeown's string matching routines. Created expand().

NAME

lstlib – routines for manipulating LST list structures

SYNOPSIS

#include "lstlib.h"

List descriptor (public fields only):

```
typedef LST * struct {
    int length;          List length excluding final NULL.
    list blklist;        List of other blocks or entities.
    int maxlen;          Allocated length excluding final NULL.
};
```

int addlst(lst,[element0,...])
```
LST lst;
pointer element0;
```

LST closelst(lst)
```
LST lst;
```

LST copylst(inlst,[maxlen])
```
LST inlst;
```

pointer getlst(lst,minpos,[&element0,...])
```
LST lst;
pointer element0;
```

LST linklst(inlst)
```
LST inlst;
```

LST newlst(element,...)
```
pointer element;
```

printlst(lst,[format])
```
LST lst;
string format;
```

int setlst(lst,minpos,[element0,...])
```
LST lst;
pointer element0;
```

LST takelst(inlst,[pos0,...])
```
LST inlst;
```

boolean validlst(lst)
```
LST lst;
```

DESCRIPTION

A LST is a header structure for a C-style list. (See listlib(3) or testbed.h for the definition. A list is essentially a vector of pointers in the manner of argv.) This header allows lists to be recognized, simplifies list length tests, and permits the list to contain NULL elements. It also has some advantages in dynamic list management, although a linked "chain" structure might be better. Public fields are listed above. They should usually be treated as read-only. References to the fields should have the form "lst->length", where lst is declared to be of type LST.

addlst(lst,[element0,...]) -> int
: Appends elements to lst and returns the resulting list length. If the original list allocation is too short, the list will be reallocated to the required length; extra cells are also allocated if the list is being extended by only one element.

closelst(lst) -> LST
: If the LST is multiply linked, simply decrement the link count; otherwise free the LST structure. (At the moment only the LST is freed, not the things that it points to.) The return value is generally NULL, but may be the original LST value if an error occurred.

copylst(inlst,[maxlen]) -> LST
: Copy all fields of the list to a new LST structure. (Only the pointers are copied, not the objects pointed at.) Then new list may be allocated to maxlen elements to allow for future growth.

getlst(lst,minpos,[&element0,...]) -> pointer
: Extract elements of lst beginning at blklist[minpos] and continuing until every argument element has been assigned a value. If the list is too short, some arguments will be left unaltered. The return value from getlst() is blklist[minpos], even if there are no elements in the argument list. This value may be treated as a string, BLK, array, or other object.

linklst(inlst) -> LST
: Link to an existing list structure. Any change in one LST will also affect the other, but they may be closed independently.

newlst(element,...) -> LST
: Create a new LST with the specified elements. Currently there is no way to specify the allocated length; it will be just large enough to hold the list elements and a final NULL.

printlst(lst,[format])
: Print the elements of a LST structure. If format is given, it should include all spaces and linefeeds for displaying one element. If it is omitted, printlist() will be called to display the list elements as an array of strings (as for a directory listing).

setlst(lst,minpos,[element0,...]) -> int
: Set elements of lst beginning at blklist[minpos] and continuing until every argument element has been used. If the list is too short, it will be reallocated to the required length; a warning message will be printed if this happens. The return value is the number of elements set; it will be

zero if lst is NULL and NOTINT if lst is invalid.

takelst(inlst,[pos0,...]) -> LST
    Construct a new LST from the specified elements of inlst.

validlst(lst) -> boolean
    Check for a non-null LST with a valid structure identification code.

FILES

/iu/tb/include/lstlib.h
/iu/tb/lib/sublib/lstlib/*
/iu/tb/lib/sublib.a

SEE ALSO

arglib(3), blklib(3), listlib(3)

DIAGNOSTICS

The lstlib routines use the printerr error reporting system. An error will typically cause a message to be printed on the error stream, and the routine will return a null LST.

HISTORY

02-Feb-83 Laws at SRI-IU
    Created this document.

NAME

matrixlib – dynamic matrices with arbitrary bounds

SYNOPSIS

```
#include <errno.h>          (optional)
#include "wdwlib.h"         (optional)
#include "matrix.h"

eigen(s,ev,need_eigenvalues)
   dmat s,ev;
   int need_eigenvalues;

eigenf(s,ev,need_eigenvalues)
   fmat s,ev;
   int need_eigenvalues;

freemat(matrix)
   imat matrix;              /* or fmat, or ... */

double matinvert(mat)
   dmat mat;

matmul(a,b,c)
   dmat a,b,c;

matmulf(a,b,c)
   fmat a,b,c;

imat newimat(lb1,ub1,lb2,ub2,error)
cmat newcmat(lb1,ub1,lb2,ub2,error)
smat newsmat(lb1,ub1,lb2,ub2,error)
lmat newlmat(lb1,ub1,lb2,ub2,error)
fmat newfmat(lb1,ub1,lb2,ub2,error)
dmat newdmat(lb1,ub1,lb2,ub2,error)
   int lb1,ub1,lb2,ub2,*error;
   imat matrix;              /* or fmat, or ...  */

printimat(matrix,colwidth)
printcmat(matrix,colwidth)
printsmat(matrix,colwidth)
printlmat(matrix,colwidth)
printfmat(matrix,colwidth,ndecimals)
printdmat(matrix,colwidth,ndecimals)
   imat matrix;              /* or fmat, or ...  */
   int colwidth,ndecimals;

readcmat(fd,hdrbytes,datawdw,pixelbytes,matrix)
readsmat(fd,hdrbytes,datawdw,pixelbytes,matrix)
readimat(fd,hdrbytes,datawdw,pixelbytes,matrix)
readlmat(fd,hdrbytes,datawdw,pixelbytes,matrix)
readfmat(fd,hdrbytes,datawdw,pixelbytes,matrix)
readdmat(fd,hdrbytes,datawdw,pixelbytes,matrix)
   int fd,hdrbytes;
   WDW datawdw;
   int pixelbytes;
   cmat matrix;             /* or fmat, or ...  */
```

rollmat(mat,r)

```
   imat mat;                /* or fmat, or ...  */
   int r;
```

DESCRIPTION

These routines and types provide runtime allocation of matrices with arbitrary lower and upper bounds. A matrix type is a small record. A sample declaration is:

```
typedef struct {
   int lb1, ub1, lb2, ub2;
   char *mat_sto;
   double **el;
} dmat;
```

Cmat is the type of a matrix of characters or bytes; *smat*, *imat*, *lmat*, *fmat*, and *dmat* represent matrices of shorts, ints, longs, floats, and doubles, respectively.

lb1 and *ub1* define the lower and upper bounds for the row index, and *lb2* and *ub2* define the lower and upper bounds for the column index. *Mat_sto* contains the address of the dynamically allocated storage. It is used by *freemat*, but should be ignored by the user.

There is an initial vector that has as many entries as there are rows of the matrix. Each entry in the vector contains the virtual base address of the corresponding row of the matrix. The value contained in *el* is the virtual base address of the vector. It is used, via normal array syntax, to access element row addresses. The element rows are stored contiguously.

The matrix record is separate from the block of dynamically allocated storage that contains the element array and initial vector. Thus, it is reasonable to pass matrices to procedures by value. The actual array elements are still passed by reference.

Eigen computes the eigenvalues and corresponding eigenvectors of the double-precision symmetric matrix *s*. *Eigenf* is identical, save that it operates on matrices of single-precision floats.

During computation, the old contents of (the diagonal and upper triangle of) *s* are destroyed, and the eigenvalues are developed in the diagonal elements. The eigenvalues are not sorted into any particular order.

If *need_eigenvalues* is true (nonzero), then the eigenvectors are calculated and placed into the rows of *ev*. The i'th row of *ev* receives the eigenvector whose eigenvalue is in the i'th diagonal position of *s*.

Matrices *s* and *ev* are as defined in the matrix(3m) package. They must be square and have identical bounds. The row bounds must equal the column bounds. The particular values of the bounds are immaterial. In particular, origin-0 and origin-1 matrices are acceptable.

S is only assumed to be symmetric; the elements in the lower triangle are never referenced.

Freemat is a macro that frees the dynamic storage (array elements and initial vector) of a matrix.

Matinvert performs in-place inversion of the matrix *mat*, and returns the determinant.

Matmul and *matmulf* perform the matrix multiplication *c := a*b* , where *a*, *b*, and *c* are matrices as defined above. *Matmul* handles double-precision matrices, while *matmulf* handles single-precision.

The matrix dimensions must be commensurate. This means that *c* and *a* must have the same number of rows; *c* and *b* must have the same number of columns; and *a* must have the same number of columns as *b* has rows. The actual index origins are immaterial. If *c* is the same matrix as *a* or *b*, the result will be correct, but processing will take more time than processing distinct matrices.

The *printmat* routines print the corresponding matrix types using *colwidth* characters per field and *ndecimals* places after the decimal point, if any. Column and row numbers are also printed in the top and left margins. These routines are only useful for small matrices.

The *readmat* routines read data into a matrix from a file. The open file *fd* is assumed to have a header of *hdrbytes* bytes followed by a data array described by *datawdw*. (See wdwlib(3) for a description of WDW structures.) The matrix coordinates should be entirely contained within the specified file window.

Rollmat effectively barrel-rolls a matrix. Row i gets rolled to become row i- *r*, etc. Shifting of the rows is upward if *r* >0, and downward if *r* <0. Rows that get shifted out the top enter at the bottom, and conversely.

Rollmat does not actually move the matrix elements around; rather, it rolls the initial vector.

Rollmat is useful for tasks such as low-pass filtering an image in place, where a few rows are buffered in a matrix.

EXAMPLE

```
#include "matrix.h"
dmat xx;
xx = newdmat(-10, 10, -15, 15, &error);
if (error) quit(1,"Out of storage.\n");
subr(xx);
freemat(xx);

subr(matrix) dmat matrix;
  {
  int i, j;
```

```
    for (i=matrix.lb1; i<=matrix.ub1; i++)
        for (j=matrix.lb2; j<=matrix.ub2; j++)
            matrix.el[i][j] = atan2( (double)i, (double)j);
    }
```

FILES

/iu/tb/include/matrix.h
/iu/tb/lib/sublib/matrixlib/*
/iu/tb/lib/sublib.a

SEE ALSO

view(1), blklib(3), vectorlib(3)

"System/360 Scientific Subroutine Package (360A-CM-03X) Version III Programmer's Manual" (IBM, 1968) contains the Fortran version from which the algebraic routines were cribbed. That source contains the following comment:

Diagonalization method originated by Jacobi and adapted by Von Neumann for large computers as found in "Mathematical Methods for Digital Computers", edited by A. Ralston and H.S. Wilf, John Wiley and Sons, New York, 1962, chapter 7.

DIAGNOSTICS

Eigen returns 0 for success, nonzero for failure. Failure cause: bad array bounds.

Matinvert: if *mat* is not square, *errno* is set to EDOM, and zero is returned. See intro(2) – for explanation of *errno*. If needed working storage cannot be obtained (only possible on a matrix larger than 100x100), *errno* is set to ENOMEM, an error message is printed, and zero is returned.

Matmul routines: if the matrix dimensions are incommensurate, *errno* is set to EDOM. If a temporary working matrix is needed (*c* is the same matrix as *a* or *b)* and cannot be obtained, *errno* is set to ENOMEM and an error message is printed.

Newmat routines: to indicate success, the int addressed by *error* will be set to zero (false). Otherwise, it will be set to nonzero (true). Error conditions are storage exhaustion, and bounds that indicate zero or fewer rows or columns.

BUGS

Bounds checking is not available.

This matrix package should be integrated with the Testbed by converting it to a blklib package. Matrices would then be passed as MTX structure pointers instead of dmat structures, etc.

HISTORY

02-Aug-83 Laws at SRI-IU
    Changed readmat() to readcmat(), readsmat(), etc.

10-Mar-83 Laws at SRI-IU
    Added the printmat and readmat routines. Changed "Iliffe vector" references to "initial vector."

11-Feb-83 Laws at SRI-IU
    Combined the separate man pages.

26-Feb-82 David Smith (drs) at Carnegie-Mellon University
Created matinvert(). Previously created rollmat(), matmul(), eigen() routines.

25-Nov-80 David Smith (drs) at Carnegie-Mellon University
Changed virtual base address name to "el" for all data types (Previously vali, vald, ...) This was possible because of the compiler enhancement, which keeps different structure declarations separate.

30-Oct-80 David Smith (drs) at Carnegie-Mellon University
New version that uses the record-containing bounds, etc.

28-Oct-80 David Smith (drs) at Carnegie-Mellon University
Created.

NAME

parselib — CMU command-line argument parsing library

SYNOPSIS

```
#include "parselib.h"

int nextbool(ptr,brk,prompt,defalt)
   char **ptr,*brk,*prompt;
   int defalt;

int nextchar(ptr,brk,prompt,legals,defalt)
   char **ptr,*brk,*prompt,*legals,defalt;

double nextdouble(ptr,brk,prompt,min,max,defalt)
   char **ptr,*brk,*prompt;
   double min,max,defalt;

float nextfloat(ptr,brk,prompt,min,max,defalt)
   char **ptr,*brk,*prompt;
   float min,max,defalt;

unsigned int nextoctal(ptr,brk,prompt,min,max,defalt)
unsigned int nexthex(ptr,brk,prompt,min,max,defalt)
   char **ptr,*brk,*prompt;
   unsigned int min,max,defalt;

int nextint(ptr,brk,prompt,min,max,default)
   char **ptr,*brk,*prompt;
   int min,max,default;

long nextlong(ptr,brk,prompt,min,max,default)
   char **ptr,*brk,*prompt;
   long min,max,default;

short nextshort(ptr,brk,prompt,min,max,default)
   char **ptr,*brk,*prompt;
   short min,max,default;

char *nextstring(ptr,brk,prompt,defalt,buffer)
   char **ptr,*brk,*prompt,*defalt,*buffer;

int nextlist(ptr,brk,prompt,optlst,defalt)
   char **ptr,*brk,*prompt,**optlst,*defalt;

int nextoption(ptr,brk,prompt,optlst,defalt)
   char **ptr,*brk,*prompt,**optlst,*defalt;
   extern char _argbreak;

char *nextarg(p,brk)
   char **p,*brk;

char *skipto(string,charset)
   char *string,*charset;
```

char *skipover(string,charset)
    char *string,*charset;

DESCRIPTION

These routines are especially useful for parsing values from argument lists in programs using the command interpreter, *ci*(3).

Nextbool attempts to parse an argument from a string, passing the string pointer *ptr* and the break character set *brk* to the *nextarg*(3) routine. If an argument is parsed, and it is a legal boolean, then its value is returned by *nextbool*. If there is no argument, or it is not valid, then an error message is printed and the remaining arguments are passed to the routine *askbool*(3). The value of *askbool* is then returned by *nextbool*.

Nextchar attempts to parse an argument from a string, passing the string pointer *ptr* and the break character set *brk* to the *nextarg*(3) routine. If an argument is parsed, and it begins with a character that is present in the string *legals* of legal character responses, then the index of that character within *legals* is returned. If there is no argument, or if it does not begin with a legal character, then the parameters *prompt*, *legals*, and *defalt* are passed to the *askchar*(3) routine, whose value is then returned by *nextchar*.

Nextdouble and *nextfloat* attempt to parse an argument from a string, passing the string pointer *ptr* and the break character seg *brk* to the *nextarg*(3) routine. If an argument is parsed, and it is a legal floating-point value between *min* and *max*, then it is returned by *nextdouble* or *nextfloat*. If there is no argument, or if it is not a valid floating-point number, or if it is out of range, then an error message is printed and the remaining parameters are passed to the routine *askdouble*(3) or *askfloat*(3); the value returned by *askdouble* or *askfloat* is then returned by *nextdouble* or *nextfloat*.

Nextoctal and *nexthex* attempt to parse an argument from a string, passing the string pointer *ptr* and the break character set *brk* to the *nextarg*(3) routine. If an argument is parsed, and it is a legal octal or hexadecimal value between *min* and *max*, then this value is returned by *nextoctal* or *nexthex*. If there is no argument, or it is not a legal value, then an error message is printed and the remaining parameters are passed into *askoctal*(3) or *askhex*(3), whose value will be returned by *nextoctal* or *nexthex*.

A legal octal value is a sequence of octal digits. A legal hexadecimal value is a sequence of digits "0" through "9", "a" through "f", or "A" through "F", and optionally preceded by the (ignored) prefix "0x" or "0X".

Nextint, *nextlong*, and *nextshort* attempt to parse an argument from a string, passing the string pointer *ptr* and the break character set *brk* to the *nextarg*(3) routine. If an argument is parsed, and it is a legal integer whose value is between *min* and *max*, then its value is returned by *nextint*, *nextlong*, or *nextshort*. If there is no argument, or it is not a valid integer, or the value is out of range, then an error message is printed and the remaining arguments are passed to *askint*(3), *asklong*(3), or *askshort*(3). The resulting value is then returned by *nextint*, *nextlong*, or *nextshort*.

Nextstring will attempt to parse an argument from a string, passing the string pointer *ptr* and the break character set *brk* to the *nextarg*(3) routine. If an argument is parsed, then it is copied into *buffer* and *nextstring* returns *buffer* as its value. If there is no argument, then *prompt*, *defalt*, and *buffer* are passed to *askstring*(3), whose value is returned by *nextstring*.

Nextlist will attempt to parse an argument in the same way. If there is an argument, and it matches exactly one string in the string array *optlst*, then the index of the matching string is returned by *nextlist*. If there is no argument, or if it matches nothing in the optlst or many entries of the optlst, then an error message is printed and the remaining arguments (*prompt*, *optlst*, and *defalt*) are passed into *asklist*(3). The value returned by *asklist* is then returned by *nextlist*.

Nextoption is just like *nextlist*, but performs a heuristic test for the best string match within the optlst, and allows the user to approve or disapprove this choice if it is not a perfect match. If the user does not approve the choice, several other good matches are listed.

Nextarg is used to parse a string, picking off one argument with each call of *nextarg*. The arguments are separated by some special "break" characters, and may additionally be surrounded by leading and trailing blanks and tabs.

When you have a string that you wish to parse, you should declare a pointer and point to the start of the string:

```
char string[100];      /* the arg list */
char *pointer;         /* a pointer */
char *argument;        /* one argument */
...
pointer = string;      /* start of arg list */
```

Then, each call to *nextarg* will fetch the next argument:

```
argument = nextarg (&pointer,"delimiters");
```

Each call to nextarg will space the pointer up to one of the delimiters or the end of the string, whichever comes first. Then, the string will be chopped into two pieces: the part containing the argument just parsed, and the rest of the string. The address of the first part will be returned; the pointer will be left pointing to the second part, all ready for the next call to nextarg. Note that the pointer must not be in a register, since it is passed by address.

The delimiter character (or null character at the end of the string) that was encountered is placed in the external variable called *_argbreak*. You may look at this value to see what delimiter was encountered. If no delimiters were encountered before the end of the string, then the null character will be placed into *_argbreak*.

If *brk*, the list of break characters, is 0, then the space character will be used as the only delimiter.

Skipto takes two arguments: *string*, a pointer to a string, and *charset*, a list (string) of characters to be searched for. *Skipto* returns a pointer to the first occurence within *string* of any of the characters in *charset*. If no such character is found in *string*, then a pointer to the null character terminating *string* is returned.

Skipover performs an identical function, but looks for the first character in *string* that does **not** occur within *charset*.

FILES

/iu/tb/include/parselib.h
/iu/tb/lib/sublib/parselib/*
/iu/tb/lib/sublib.a

SEE ALSO

arglib(3), index(3), ci(3), scanf(3)

DIAGNOSTICS

If the argument is null, or the end of the string is reached, then a pointer to the null string will be returned. At the end of the string, nextarg() may be repeated any number of times – it will leave the pointer unchanged, and will return a pointer to the null string.

BUGS

For most purposes it is better to use the Testbed arglib routines.

HISTORY

01-Feb-83 Laws at SRI-IU
Changed the ...arg() names to next...() form to remove conflicts with the Testbed arglib.

17-Nov-82 Laws at SRI-IU
Combined all of the CMU arglib **man** pages into this file.

21-Dec-81 Laws at SRI-IU
Changed names from chrarg to chararg, octarg to octalarg, strarg to stringarg, stabarg to listarg, searcharg to optionarg, and nxtarg to nextarg.

23-Jan-80 Steven Shafer (sas) at Carnegie-Mellon University
Searcharg added.

05-Dec-79 Steven Shafer (sas) at Carnegie-Mellon University
Created.

NAME

piciolib – low-level pixel access library

SYNOPSIS

#include "piclib.h"

Unsigned integer pixel routines:

```
getbox(pic,mincol,maxcol,minrow,maxrow,buff,bufcols)
    PIC pic;
    int *buff;

getcol(pic,col,minrow,maxrow,buff)
    PIC pic;
    int *buff;

getrow(pic,row,mincol,maxcol,buff)
    PIC pic;
    int *buff;

int getpixel(pic,col,row)
    PIC pic;

putbox(pic,mincol,maxcol,minrow,maxrow,buff,bufcols)
    PIC pic;
    int *buff;

putcol(pic,col,minrow,maxrow,buff)
    PIC pic;
    int *buff;

putpixel(pic,col,row,value)
    PIC pic;

putrow(pic,row,mincol,maxcol,buff)
    PIC pic;
    int *buff;
```

Special packing/unpacking routines:

```
g_getbox(pic,mincol,maxcol,minrow,maxrow,buff,shift)
    PIC pic;
    char *buff;

g_putbox(pic,mincol,maxcol,minrow,maxrow,buff,shift)
    PIC pic;
    char *buff;

getpkbx(pic,mincol,maxcol,minrow,maxrow,buff,bufcols)
    PIC pic;
    char *buff;
```

DESCRIPTION

A PIC is the handle (or "capability") by which one data band of a picture file is accessed. The PIC must be passed to each access routine so that the proper paging mechanism can be used. See piclib(3) for ways to obtain this valuable object.

The vector routines require a buffer address, which may be an integer array or a pointer into such an array. The block routines require both a pointer to the first affected element and the number of columns in the array. The implied subscript ranges are generally checked against the physical data range in the PIC.

The column and row addressing is relative to (0,0) in the lower left corner of the imaged scene, regardless of the scan pattern used to store the image data. (The picture dimensions in the PIC structure have similarly been corrected for scan pattern.)

Unsigned integer pixel manipulation:

getbox(pic,mincol,maxcol,minrow,maxrow,buff,bufcols)
: Unpacks a picture window into a matrix window.

getcol(pic,col,minrow,maxrow,buff)
: Unpacks a partial column of image pixels into a buffer.

getrow(pic,row,mincol,maxcol,buff)
: Unpacks a partial row of image pixels into a buffer.

getpixel(pic,col,row) -> int
: Return the value of the indicated pixel. The pixel bit pattern is assumed to match the pixeltype specified in the PIC; no conversion (other than extension to 32 bits) is done.

putbox(pic,mincol,maxcol,minrow,maxrow,buff,bufcols)
: Packs a matrix window into a picture window.

putcol(pic,col,minrow,maxrow,buff)
: Packs a vector of pixels into a picture column.

putpixel(pic,col,row,value)
: Set the value of the indicated pixel. There is no checking to see if the value is legal for this picture file.

putrow(pic,row,mincol,maxcol,buff)
: Packs a vector of pixels into a picture row.

Special packing/unpacking routines:

g_getbox(pic,mincol,maxcol,minrow,maxrow,buff,shift)

g_putbox(pic,mincol,maxcol,minrow,maxrow,buff,shift)
: These are analogous to the unsigned pixel routines, but pack the data into byte (char) buffers or matrices as required by the Grinnell display. The routines leave no pad space between the rows of "buff," and each pair of pixels within "buff" is in swapped order. Each pixel may be left-shifted (scaled) by "shift" positions during the packing. It is expected that users will not need to call these procedures directly.

getpkbx(pic,mincol,maxcol,minrow,maxrow,buff,bufcols)
: Unpacks a 1-bit picture window into an 8-bit matrix window.

FILES

/iu/tb/include/piclib.h
/iu/tb/lib/imagelib/piciolib/*
/iu/tb/lib/imagelib.a

A similar set of routines is in /iu/tb/lib/cmuimglib/piciolib. They use the CMU IMAGE structure instead of the Testbed PIC structure.

SEE ALSO

cmuimglib(3), gmrlib(3), hdrlib(3), piclib(3)

DIAGNOSTICS

The piciolib routines generally cannot detect an invalid PIC argument, which will cause a core dump. A bounds error will usually trigger an error message and an exit to system level. An out-of-bounds getpixel() request will exit without an error message.

BUGS

The "g_" names in piciolib are incompatible with MAINSAIL interfacing.

The Testbed routines currently support only single-band, unsigned pixels of 1 to 32 bits, although the Testbed header permits floating-point and other pixel formats.

Signed-pixel routines identical to the unsigned-pixel routines listed above were contributed by CMU. They will be made available if needed.

These routines are currently implemented in assembly language, which presumably makes them fast but also makes them difficult to maintain or alter.

The getbox() and putbox() commands are currently implemented using repeated calls to getrow() and putrow(). This may not be optimal for image data stored in block raster format, as is common on the Testbed.

If the number of pixels in the window ((erow-srow+1)*(ecol-scol+1)) is odd, then g_getbox() and g_putbox() pad the buffer with a noise byte, the contents of which are not guaranteed. G_putbox() expects the unused bits of each byte in the buffer to be zero.

UNIX write() calls to the pic->fd may not work because the file is valloc'ed. Direct access to the image data (or any header or trailer records) may require that the file be reopened using the ordinary open() call.

See the source directory for additional comments.

HISTORY

25-Feb-83 Laws at SRI-IU
: Changed buff for the special packing routines from unsigned short int to char. This conforms to the CMU man page, although it may conflict with a comment in the assembly code header.

26-Nov-82 Laws at SRI-IU
: Changed setpixel() to putpixel() and pkbx() to getpkbx(), renamed various column and row arguments, and reordered the arguments so that columns are specified before rows.

10-Nov-82 Laws at SRI-IU
: Created this document. The iolib routines are adapted from those contributed to the Testbed by Carnegie-Mellon University. Pixel() has been changed to getpixel().

04-Mar-80 David Smith (drs) at Carnegie-Mellon University
: Created g_getbox() and g_putbox().

NAME

piclib – single-band picture access library

SYNOPSIS

#include "piclib.h"

Picture descriptor (public fields only):

```
typedef PIC * struct {
   string filename;            Picture file name.
   int ncols;                  Pixels across (after scan conversion).
   int nrows;                  Pixels down (after scan conversion).
   int pixelbits;              Physical bits per pixel.
   BANDTYPE pixeltype;         Pixel format code.
   HDR hdr;                    Picture HDR structure.
   DOC doc;                    Picture documentation structure.
   boolean writedoc;           Write documentation during closing.
};
```

clippic(inpic,inwdw,outpic,minthresh,minval,maxthresh,maxval,elseval)
```
   PIC inpic,outpic;
   WDW inwdw;
```

PIC closepic(pic)
```
   PIC pic;
```

PIC copypic(inpic,outname)
```
   PIC inpic;
   string outname;
```

copypicwdw(inpic,inwdw,outpic,outwdw)
```
   PIC inpic,outpic;
   WDW inwdw,outwdw;
```

PIC linkpic(inpic)
```
   PIC inpic;
```

PIC newpic([hdr,filename,protection])
```
   HDR hdr;
   string filename;
```

PIC openpic([filename,mode])
```
   string filename;
```

PIC openrawpic(hdr,filename,mode,creating)
```
   HDR hdr;
   string filename;
   boolean creating;
```

DOC opnpicdoc(filename)
```
   string filename;
```

int picmode(pic)
  PIC pic;

printpic(pic,[verbose])
  PIC pic;
  boolean verbose;

PIC scrappic(pic)
  PIC pic;

setsoftpic(kbytes)

PIC tmppic([hdr])
  HDR hdr;

boolean validpic(pic)
  PIC pic;

DESCRIPTION

A PIC is the handle (or "capability") by which one data band of a picture file is accessed. Public fields are listed above. They should usually be treated as read-only. References to the fields should have the form "pic->ncols," where pic is declared to be of type PIC.

The public information in the PIC is essentially copied from the picture file header, as represented by the associated HDR. The picture dimensions, ncols and nrows, have been corrected for the scan pattern so that they represent scene dimensions rather than data storage dimensions. All picture coordinates are zero-based, beginning with (0,0) in the lower left corner. For pixeltype codes, see hdrlib(3) or the Testbed picture header documentation.

clippic(inpic,inwdw,outpic,minthresh,minval,maxthresh,maxval,elseval)

Copy a window of data from inpic into outpic, clipping the gray levels below minthresh and above maxthresh. Clipped pixels will be mapped to minval or to maxval, respectively. Remaining pixels will be copied as is if elseval is NOTINT; otherwise they will be set to elseval. If inwdw is NULL, the entire input image will be copied; otherwise only the specified data window will be. [See wdwlib(3) for methods of creating WDWs.] Outpic must be the same size as the requested window.

closepic(pic) -> PIC

If the PIC is multiply linked, simply decrement the link count. Otherwise update and close the picture file and free the PIC structure. Writeable picture files that are not closed will usually be truncated. The return value is generally NULL, but may be the original PIC value if an error occurred.

copypic(inpic,outname) -> PIC

Copy all pixel data and paging mechanisms associated with inpic to a new file and PIC. The two PICs will then be entirely separate.

copypicwdw(inpic,inwdw,outpic,outwdw)

Copy a window of data from inpic into a window in outpic; at present the two windows must be the same size. See wdwlib(3) for methods of creating WDWs. If you find picture windows useful, you might want to investigate imglib(3).

linkpic(inpic) -> PIC

Copy the original PIC pointer and increment the link count of the associated data structure. The new PIC will now reference the same paging structure. Closing either PIC will simply break the link. Both PICs must be closed to actually close (and save) the file.

newpic([hdr,filename,protection]) -> PIC

Create a new PIC with the specified header, file name, and protection mode (as for the UNIX creat command), then open it for RW access. The hdr argument may be taken from another PIC, created using newhdr() (see hdrlib(3)), or omitted or left NULL if the user is to be asked for the required information. The filename may also be omitted or NULL if the user is to be asked for a name. The default protection mode is 0664, which stands for rw-rw-r--. A NODOC documentation structure will be created when the picture is opened. No documentation record will be written by closepic() unless the doc->doctype is changed and pic->writedoc is set to TRUE.

openpic([filename,mode]) -> PIC

Open the specified file for picture access; the PICPATH environment variable (default ":/iu/tb/pic:/aux/tbpic") is searched if necessary. The mode (0 => R, 1 => W, 2 => RW) is the same as for the UNIX open() command; the default is 0. The picture will be opened for hardware paging (virtual address mapping) or for software paging, depending on the setsoftpic() threshold and the amount of virtual address space available. A documentation structure, pic->doc, will be created to hold the documentation record associated with this file; it will have doctype NODOC if there is currently no documentation. The pic->writedoc boolean will be set to FALSE.

openrawpic(hdr,filename,mode,creating) -> PIC

Open the specified file for picture access using the supplied header. This can be useful for overriding the stored file header (e.g., to read a multiband file as if there were no band structure) or for bypassing error checking in the header parsing routines. The "creating" flag is used to set a bit in the output PIC that indicates whether buffered data pages must be written out during file closing.

opnpicdoc(filename) -> DOC

Obtain a documentation structure containing any descriptive text for the picture file; see doclib(3) for details. The standard directory path is searched for the specified filename. This routine is only needed if you do not want to open the picture file for pixel I/O.

The documentation text in pic->doc will be written when closing the file if pic->writedoc is TRUE. Be sure to set this boolean if you make permanent changes to pic->doc->doctext.

picmode(pic) -> int
: Return the system protection mode associated with the image file. Returns 0664 and prints a warning if the mode cannot be determined. (A printerr WARNING status is also returned.)

printpic(pic,[verbose])
: Print partial contents of a PIC structure or a list of PICs. This is primarily used for debugging, although it may be of use in verifying to the user the success of opening a picture file. The optional verbose causes printing off all paging status fields in the PIC structure.

scrappic(pic) -> PIC
: If the PIC is multiply linked, simply decrement the link count. Otherwise delete the picture file and free the PIC structure. The return value is generally NULL, but may be the original PIC if an error occurred.

setsoftpic(kbytes)
: Set the software paging threshold to this image size (in 1024-byte pages). Smaller images will be opened with hardware paging (virtual address mapping) if sufficient virtual address space remains; larger ones will be copied into memory page by page, as required. This has no effect on pictures that are currently open. The default is setsoftpic(INF), or always try for hardware paging.

tmppic([hdr]) -> PIC
: Create a temporary PIC with the specified header, then open it for RW access. The filename will be "", and the file itself will be "unlinked" from the UNIX file system. This means that it will disappear if the picture is closed or if the process is terminated.

validpic(pic) -> boolean
: Check for a non-null PIC with a valid structure identification code. This test should be applied to a PIC before using it for pixel access.

EXAMPLE

For simple example of usage, see demo/copypic.c and demo/showpic.c in /iu/tb/lib/imagelib. Similar Testbed routines with full argument parsing and error checking are invert.c and show.c in corresponding subdirectories of /iu/tb/src.

FILES

/iu/tb/include/piclib.h
/iu/tb/lib/imagelib/piclib/*
/iu/tb/lib/imagelib/piciolib/page.c for setsoftpic()
/iu/tb/lib/imagelib.a
/iu/tb/lib/sublib.a
The math library (-lm) is also required.

SEE ALSO

blklib(3), doclib(3), hdrlib(3), imgfrmlib(3), imglib(3), piciolib(3)

DIAGNOSTICS

The piclib routines use the printerr error reporting system. An error will typically cause a message to be printed on the error stream and the routine will return a null PIC. If given a null PIC or HDR as an argument, the opening routines will generally ask for the needed information.

BUGS

Openpic() currently fails for data scanned vertically.

The tmppic() routine stores "" as the file name of a temporary picture. Perhaps the pointer should be set to NULL instead. (Printpic() and possibly other routines would have to check for this.)

If you change the associated HDR structure, you should also set pic->writeheader to TRUE. This will cause the new information to be written out when the file is closed. Perhaps this field should be moved to the public (i.e., documented) portion of the PIC.

UNIX write() calls to the pic->fd may not work because the file is valloc'ed. Direct access to the image data (or any header or trailer records) may require that the file be reopened using the ordinary open() call.

HISTORY

27-Sep-83 Laws at SRI-IU
Added picmode().

06-Sep-83 Laws at SRI-IU
Added opnpicdoc() and the DOC information.

06-Dec-82 Laws at SRI-IU
Made the hdr argument to tmppic() optional.

20-Nov-82 Laws at SRI-IU
Changed imgsoft() to setsoftpic() and cpypicwdw() to copypicwdw(). Made the mode arguments to openpic() and newpic() optional.

10-Nov-82 Laws at SRI-IU
Created this document.

NAME

pntlib – routines for manipulating point structures

SYNOPSIS

#include "pntlib.h"

Point descriptor (public fields only):

```
   typedef PNT * struct {

      float x;          Abscissa.
      float y;          Ordinate.
      int col;          Rounded or truncated x.
      int row;          Rounded or truncated y.
   };
```

PNT closepnt(pnt)
   PNT pnt;

PNT copypnt(inpnt)
   PNT inpnt;

PNT linkpnt(inpnt)
   PNT inpnt;

PNT newintpnt(col,row)

PNT newpnt(x,y)
   float x,y;

PNT pntarg(cmdarg,limitwdw,dftpnt,[askmsg])
  WDW limitwdw;
  PNT dftpnt;
  string askmsg;

printpnt(pnt)
   PNT pnt;

boolean validpnt(pnt)
   PNT pnt;

DESCRIPTION

A PNT is an abstract grouping of coordinates to represent a two-dimensional position in an image, picture file, or array. Public fields are listed above. They should usually be treated as read-only. References to the fields should have the form "pnt->x", where pnt is declared to be of type PNT.

closepnt(pnt) -> PNT

If the PNT is multiply linked, simply decrement the link count; otherwise free the PNT structure. The return value is generally NULL, but may be the original PNT value if an error occurred.

copypnt(inpnt) -> PNT
: Copy all fields of the point to a new PNT structure. The two PNTs will then be entirely separate.

linkpnt(inpnt) -> PNT
: Link to an existing point structure. Any change to one PNT will also affect the other, but they may be closed independently.

newintpnt(col,row,[roundmode]) -> PNT
: Create a new PNT with the specified integer coordinates. The corresponding floating-point coordinates will be generated according to roundmode:

'd'	– round down (floor)
'n'	– round to nearest integer (default)
'u'	– round up (ceiling).

newpnt(x,y) -> PNT
: Create a new PNT with the specified floating-point coordinates. The corresponding integer coordinates will be generated. (There is currently no test for overflow.)

pntarg(cmdarg,limitwdw,dftpnt,[askmsg]) -> PNT
: Interactively obtain a PNT specification. Limitwdw may be NULL if there is no limit on the point range; dftpnt may be NULL if there is no default. Askmsg may be NULL, a string, or a LST of prompt strings (initial prompt, point prompt, and additional help message). See arglib(3) for further details.

printpnt(pnt)
: Print partial contents of a PNT structure. This is primarily used for debugging.

validpnt(pnt) -> boolean
: Check for a non-null PNT with a valid structure identification code. This does not currently check for consistency of the floating-point and integer coordinates.

FILES

/iu/tb/include/pntlib.h
/iu/tb/lib/sublib/pntlib/*
/iu/tb/lib/sublib.a

SEE ALSO

arglib(3), blklib(3), seglib(3), wdwlib(3)

Crsarg() in gmrlib may also be used to obtain a PNT value, either from the keyboard or from a cursor. It is discussed in arglib(3).

DIAGNOSTICS

The pntlib routines use the printerr error reporting system. An error will typically cause a message to be printed on the error stream and the routine will return a null PNT.

BUGS

Do not try to store infinities. FLOATINF is much larger than INF, so that compatibility is a problem. (There are currently no checks for such overflow problems.) It is usually cleaner to use a NULL point than an infinite one.

HISTORY

02-Feb-83 Laws at SRI-IU
Created this document.

NAME

printerr – error reporting package

SYNOPSIS

#include "err.h"

Status variables:
  int errstatus;
  int localstatus;

Formatting variables:
  string errskip;
  string errindent;
  string errprefix[];
  string errsuffix[];
  string buglabel;

Status control routines:
  errname(string);
  seterrflags(error,quit,warning,ok);
  restoreflags();
  seterr(newstatus);
  noteerr();

Global status queries:
  worsethan(errcode)
  statusis(errcode)
  betterthan(errcode)

Error reporting routines (values are optional):
  printerror(format,values ...);
  printquit(format,values ...);
  printwarning(format,values ...);
  printok(format,values ...);

Unconditional printing routines (values are optional):
  printargs(format,values ...);
  printbug(format,values ...);

Utility routines:

checkerr();	Check subroutine entry status.
errprint();	Print an error message.
string errstring();	Error code to ASCII name.

DESCRIPTION

Printerr is a package of macros and subroutines permitting flexible reporting of error conditions. For each "participating" routine it maintains a public status variable (errstatus) and a private status variable (localstatus). The public variable communicates error conditions from invoked routines to their callers. The private variable makes it possible to "remember" an error condition while calling other routines that also use the global variable.

The first code line of a participating program or subroutine should call errname() to declare the local status variable and save a copy of the routine name (or other identifier). Errname() also calls checkerr() to print the declared name on the standard error output if OK-level messages are enabled; this is useful for following flow of control during debugging.

The private status variable, localstatus, is originally set to OK. It keeps track of the worst reportable error encountered within this routine. A reportable error is one for which seterr(), noteerr(), or an error printing routine is called. It may or may not be reported to the user or to the calling routine, as explained below.

The public variable, errstatus, returns the completion status of subroutines, and also returns the status of the current routine when it is finished. Errstatus is set to OK on entry. (Note that this replaces the global status of the calling routine, but leaves its localstatus intact.) Thereafter errstatus is only altered by calling participating subroutines or the routines in this package.

Every time a participating subroutine is called, the global error status variable will be set to the returned status of that routine. If you ignore the status, it is implicitly accepted as the current status. This is risky since the error status may be reduced to a less severe condition by the next subroutine called; it is safer to examine the returned status and to set an appropriate error status for the current routine.

To examine the global status, use the status query macros. The conventional values (available through testbed.h) are ERROR for serious errors, QUIT for propagating previously reported errors, WARNING for non-terminal conditions, and OK for debugging messages; see below for more detail. The test "if worsethan(WARNING) ..." will tell you whether a serious error has occurred. If this is not sufficiently flexible, you may test the global or local status variables directly.

To alter the status or make it more permanent, call seterr(), noteerr(), or one of the error printing routines. Seterr() sets both the local status and the global status to its argument code. Noteerr() sets both to the min (i.e., worst condition) of the existing local status and global errstatus(). Thus seterr() gives full control over the current status, while noteerr() is useful for keeping track of the worst reportable error so far.

Seterr() should only be used by a routine that understands the source of error and knows what to do with it. Neither seterr() nor noteerr() takes any corrective action. It is more common to test the global status, report the error, and either exit, return, or correct the error condition. The error reporting routines simplify this process.

Each error reporting routine sets both status variables to the min of its argument value and the current local error. It also prints a message if the appropriate flag is set. Initially only the ERROR and WARNING flags are set; you may alter this by calling seterrflag() with a TRUE of FALSE argument to enable or disable the corresponding level of reporting. To enable all messages, add "seterrflags(TRUE,TRUE,TRUE,TRUE)" to your main program (or elsewhere). Seterrflags() also saves the previous flag values so that a call to restoreflags() can reinstate them.

Printerror() is usually called whenever a low-level (or non-participating) error condition is first discovered. This is followed by error-handling code (e.g., closing files and freeing structures) and by a return or exit statement. The argument to a return statement may be an error code, a null pointer, or any other useful value matching the declared subroutine type. An exit, which terminates the entire job, should usually return ERROR.

Printquit() is used by routines as they pass a previously-reported ERROR condition up to to be handled at a higher level. A routine receiving an ERROR or QUIT status from a subroutine should either handle and reset the error condition or pass it up as another QUIT. QUIT may also used when acceeding to an EOF or interactive abort request if it is assumed that the user knows what is happening and doesn't want to see any ERROR messages. (QUIT messages are usually suppressed by the default setting of errflags.)

Printwarning() and printok() note non-fatal status conditions. WARNING messages are advisory, and OK messages are for debugging. These are non-fatal, and will be printed with a less conspicuous format than ERROR messages. OK messages are usually suppressed.

Printargs() and printbug() are two additional print routines that prepend the routine name and either "args:" or "bug:" to the format argument that you supply. These routines do not alter the current error status, nor do they test the error status before printing.

Error messages are written to the standard output stream. Normally the print routine prints a blank line, an indentation, a prefix string, the routine name, the error message, and a suffix string. The prefix and suffix strings are chosen to make the various error levels distinguishable. You may change the error formats by changing the values of the formatting variables; see the printerr.c code for an example.

EXAMPLE

```
#include "err.h"

float scale(factor)
  float factor;
{

  float value;

  /* Initialize the error package. */
  errname("factor");

  /* Check for an unusual argument. */
  if (fabs(factor) > 1.0) then printwarning("Scale factor is %g.",factor);

  /* Ask the user for a value to be scaled. */
  value = floatarg(NOTINT,-INF,INF,0.0,"Enter a floating-point value:");

  /* Check for an error in floatarg(). */
  if worsethan(WARNING) then {
    printquit("No value specified.");
    return(0.0);
  }

  /* Scale the specified value. */
  return(factor*value);
}
```

FILES

/iu/tb/include/err.h
/iu/tb/lib/sublib/errlib/printerr.c
/iu/tb/lib/sublib.a

The following global variables are also used:

```
   string erroutine;
   int errformat;
```

SEE ALSO

quit(3)

DIAGNOSTICS

Errname() is restricted to be the first statement of a routine or block because it includes declarations. You will get strange compiler messages if you try to put it elsewhere.

BUGS

Perhaps the include file should be named printerr.h.

The "printquit()" syntax is inferior to "printerr(QUIT,...)", which would allow one to call printerr without first testing and branching on the error condition. The latter syntax could not be implemented given current C restrictions on macros with variable numbers of arguments. Perhaps it could be done using functions and the undocumented Berkeley nargs() routine.

There is no way (short of a separate preprocessor) for constructing a single multiaction "returnerr" macro that will substitute directly for C return statements. Thus user-visible ugliness would be required to completely encapsulate routines with entry and exit hooks. (This would be required, e.g., if a global depth variable were to be kept for indenting error messages according to depth of invocation. At present you can only do this by providing your own depth-dependent indentation using the errindent formatting variable.)

The errname() macro could be allowed extra arguments denoting the importance or indentation level of the current routine. These might be useful in formatting error messages.

A far more flexible dynamic tracing package could be built around the errname() capability.

The directory containing the printerr routines also contains error handlers imported from other systems. They do not have a consistent style.

HISTORY

12-Jan-84 laws at SRI-IU
Added the printargs and printbug routines.

11-Aug-82 laws at SRI-IU

Created.

NAME

putenv – put value into environment

SYNOPSIS

int putenv (name,value);
  char *name,*value;

DESCRIPTION

Putenv assigns a value to an environment parameter.

If *value* is non-zero, then a string of the form
    name=value
is added to your environment. If *name* already exists in your environment, it becomes associated with the new *value*. If not, a new *name* is added to your environment.

If *value* is 0, then *name* is deleted from your environment (if it exists).

FILES

/iu/tb/lib/sublib/syslib/putenv.c
/iu/tb/lib/sublib.a

SEE ALSO

getenv(3)

DIAGNOSTICS

Returns -1 on error (usually after an error in *malloc*(3)). Otherwise, 0 is returned.

BUGS

The first time you call *putenv*, a copy of your entire environment is created with *malloc*(3). This is painful but necessary; usually, your environment is small enough that it's not a real problem.

HISTORY

05-Dec-79 Steven Shafer (sas) at Carnegie-Mellon University
    Created.

NAME

quit – print message and exit

SYNOPSIS

quit (status, format [, arg] ...)
  int status;
  char *format;

DESCRIPTION

Quit is a means of terminating a process with an error message.

This call should never return.

Quit prints on standard error the message specified by the *printf*(3) argument list *format* [, *arg*]... then exits.

The first argument, *status*, will be the argument passed to *exit*(2) and this is the process's return code. All of the process's files will be closed.

FILES

/iu/tb/lib/sublib/errlib/quit.c
/iu/tb/lib/sublib.a

SEE ALSO

printerr(3), printf(3), exit(2), wait(2)

BUGS

The implementation of *quit* depends on a routine named *_doprnt*. Although *fprintf* and *printf* on Vax and PDP11 Unix are implemented with this routine, other Unix systems might not implement them this way.

HISTORY

20-Mar-81 Dale Moore (dwm) at Carnegie-Mellon University
  Changed to call _doprnt instead of fprintf.

06-Dec-79 Steven Shafer (sas) at Carnegie-Mellon University
  Created.

NAME

run, runv, runp, runvp – execute process and wait for it

SYNOPSIS

int run (file,arg0,arg1,arg2,...,argn,0)
int runv (file,arglist)
int runp (file,arg0,arg1,arg2,...,argn,0)
int runvp (file,arglist)
char *file,*arg1,*arg2,...,*argn,**arglist;

DESCRIPTION

Run and *runv* have argument lists identical to the corresponding functions, *execl*(2) and *execp*(2). The run routines perform a *vfork*(2), then:

In the new process:
setgid(2) and *setuid*(2) are used to ensure that privileges unique to the parent do not propagate to the child. An *execl* or *execv* is then performed with the specified arguments. The process returns with a -1 code if the *exec* was not successful.

In the parent process:
the signals *SIGQUIT* (see *signal*(2)) and *SIGINT* are disabled, the process *waits* (see *wait*(2)) until the newly forked process exits, the signals are restored to their original status, and the return status of the process is analyzed.

Run and *runv* return -1 if the exec failed or the child was terminated by a *signal;* the *exit* code of the process otherwise.

Runp and *runvp* are identical to *run* and *runv*, but perform path searching for the process by using *execlp* and *execvp*. These routines use the PATH environment parameter as a list of directory names separated by colons; the executable file is sought in each directory until it is found or all directories have been searched. If the file is not found, -1 is returned.

The proper way to execute system programs is via *runp* or *runvp* for most purposes; for example, if you want to move file "a" to "b", the best way to do this via system programs is this:

```
        runp ("mv","mv","a","b",0);
```

Note that no directory name is needed along with the name of the file (e.g. "/bin/mv" is not necessary), and that the program name should be both *file* and *arg0*. This call is similar to:

```
        system ("mv a b");
```

but is much faster to execute.

The use of *setgid* and *setuid* means that, if the parent process gained privileges through the use of special file mode bits (see *chmod*(2)), the child process will not inherit these privileges. This makes *run* "safe" for system programs which require special privileges, and usually has no effect on user programs.

ENVIRONMENT

The *PATH* environment parameter is used to find executable files in *runp* and *runvp.*

FILES

/iu/tb/lib/sublib/syslib/run.c
/iu/tb/lib/sublib/syslib/runv.c

/iu/tb/lib/sublib.a

SEE ALSO

exec(2), vfork(2), signal(2), system(3), searchp(3)

DIAGNOSTICS

These routines return -1 if any error occurs in executing the desired program. If the program is executed successfully, convention dictates that it should return 0 on normal completion and non-zero (1, 2, etc.) if any error is encountered.

BUGS

The searching rule used by *execlp* and *execvp* is not the same as the rule used by *searchp*. See the comments in *searchp*(3) for more detailed information.

HISTORY

28-Jan-80 Steven Shafer (sas) at Carnegie-Mellon University

Added setuid and setgid feature, so that run may be used by privileged programs (e.g. post) to execute programs which allow users to fork shells (e.g. text editors).

21-Jan-80 Steven Shafer (sas) at Carnegie-Mellon University

Changed fork() to vfork(). This wins speed if run (etc.) is called from inside a very large program.

05-Dec-79 Steven Shafer (sas) at Carnegie-Mellon University

Created.

NAME

seglib – routines for manipulating line segment structures

SYNOPSIS

#include "seglib.h"

Window descriptor (public fields only):

```
typedef SEG * struct {

    float x0;           First abscissa.
    float y0;           First ordinate.
    float x1;           Second abscissa.
    float y1;           Second ordinate.

    float dx;           x1-x0.
    float dy;           y1-y0.
    float length;       sqrt(dx*dx+dy*dy).
    float angle;        Angle in radians. (0 to 2PI?)
};
```

SEG clipseg(inseg,limitwdw)
SEG inseg;
WDW limitwdw;

SEG closeseg(seg)
SEG seg;

SEG copyseg(inseg)
SEG inseg;

SEG linkseg(inseg)
SEG inseg;

SEG newintseg(col0,row0,col1,row1)

SEG newseg(x0,y0,x1,y1)
float x0,y0,x1,y1;

printseg(seg)
SEG seg;

boolean validseg(seg)
SEG seg;

DESCRIPTION

A SEG is a representation of a line segment in a floating-point coordinate space. Public fields are listed above: they should usually be treated as read-only. References to the fields should have the form "seg->x0", where seg is declared to be of type SEG.

clipseg(inseg,limitwdw) -> SEG
: Return a new segment which contains the inseg coordinates clipped to the limitwdw ranges.

closeseg(seg) -> SEG
: If the SEG is multiply linked, simply decrement the link count; otherwise free the SEG structure. The return value is generally NULL, but may be the original SEG value if an error occurred.

copyseg(inseg) -> SEG
: Copy all fields of the segment to a new SEG structure. The two SEGs will then be entirely separate.

linkseg(inseg) -> SEG
: Link to an existing segment structure. Any change to one SEG will also affect the other, but they may be closed independently.

newintseg(col0,row0,col1,row1) -> SEG
: Create a new SEG with the specified integer coordinates.

newseg(x0,y0,x1,y1) -> SEG
: Create a new SEG with the specified floating-point coordinates.

printseg(seg)
: Print contents of a SEG structure. This is primarily used for debugging.

validseg(seg) -> boolean
: Check for a non-null SEG with a valid structure identification code.

FILES

/iu/tb/include/seglib.h
/iu/tb/lib/sublib/seglib/*
/iu/tb/lib/sublib.a

SEE ALSO

blklib(3), pntlib(3), wdwlib(3)

DIAGNOSTICS

The seglib routines use the printerr error reporting system. An error will typically cause a message to be printed on the error stream and the routine will return a null SEG.

BUGS

The segment angle is not currently computed.

Do not try to store infinities. A horizontal or vertical extent from -FLOATINF to FLOATINF has a range too large to be represented by dx, dy, or length. There are currently no checks for such overflow problems, although the length fields could be set to NOTFLOAT when such problems are found. It is usually cleaner to use a NULL segment than an infinite one, although representing one that is infinite in only one dimension is a problem.

HISTORY

02-Feb-83 Laws at SRI-IU
Created this document.

NAME

smoothlib – image smoothing routines

SYNOPSIS

#include "image.h"

dmedian(ipic,opic,srow,erow,scol,ecol,verbose)
dsmooth(ipic,opic,srow,erow,scol,ecol,verbose)
median_smooth(ipic,opic,window,srow,erow,scol,ecol,verbose)
nagao(ipic,opic,srow,erow,scol,ecol,verbose)
tomita(ipic,opic,srow,erow,scol,ecol,verbose)

```
IMAGE *ipic,*opic;
int window,srow,erow,scol,ecol,verbose;
```

DESCRIPTION

These subroutines use various algorithms to smooth images. *Ipic* and *opic* are pointers to images created or opened by the caller. They may be identical. The smoothing operator is applied to *ipic*, and the result is writen into *opic*. *Window* specifies the size (length of side) of the smoothing window. This must be an odd integer no less than 3, unless it is zero, which signifies the default value (3). *Srow* and *erow* specify the starting and ending rows of the subimage which are to be processed; *scol* and *ecol* bound the columns. No pixels outside of the specified range will be read from *ipic*. All pixels in the specified range will be written into *opic*. Near the borders, the smoothing window is cropped. If *verbose* is true, progress marks are printed.

The algorithms applied are as follows.

dmedian
: (Diminished median) The 3x3 neighborhood of a pixel is examined. The three pixels whose values are farthest from the central pixel are discarded. The result is the median of the remaining six values (by averaging the middle pair).

dsmooth
: is an algorithm devised by Smith which borrows from the ideas of Tomita and Nagao. If the 3x3 neighborhood of a pixel is

```
a b c
d e f
g h i
```

then the neighborhoods abcdef, abcefi, bcefhi, etc. are examined. The one with the lowest variance is selected, and its average is the resultant value.

median_smooth
: The result is the median of the pixels in the smoothing neighborhood.

nagao
: implements Nagao & Matsuyama's algorithm, reported in *Edge Preserving Smoothing*, IJCPR-78, p.518-520; also in CGIP 9, 394-407 (1979).

tomita
: is the algorithm of Tomita & Tsuji, implemented with 3x3 windows. See *Extraction of Multiple Regions by Smoothing in Selected Neighborhoods*, IEEE Trans. Syst., Man, & Cybernetics, SMC-7, 1977, p.107-109.

FILES

/iu/tb/include/image.h
/iu/tb/lib/cmuvsnlib/smoothlib/*

/iu/tb/lib/cmuvsnlib.a

SEE ALSO

reduce(1), imglib(3)

DIAGNOSTICS

median complains if the window size is inappropriate.

BUGS

No validity checking is performed on the image pointers or the row and column range specifiers.

These routines use CMU IMAGE-based image I/O. For an example of a PIC-based system, see /iu/tb/lib/visionlib/convlib. (The smoothlib routines will be converted to PIC form as needed.)

HISTORY

09-Mar-81 David Smith (drs) at Carnegie-Mellon University
Created.

NAME

stringlib – routines for manipulating C strings

SYNOPSIS

#include "stringlib.h"

string fold(outstr,instr,whichway)
```
   char outstr[];
   string instr;
   enum {FOLDUP,FOLDDOWN} whichway;
```

string folddown(outstr,instr)
```
   char outstr[];
   string instr;
```

string foldup(outstr,instr)
```
   char outstr[];
   string instr;
```

boolean leftmatch(big,small)
```
   string big,small;
```

path(filename,direc,file)
```
   string filename;
   char direc[],file[];
```

int rootcomp(filename0,filename1)
```
   string *filename0,*filename1;
```

int srchscore(big,small)
```
   string big,small;
```

string strcopy(instr)
```
   string instr;
```

unsigned int strhex(instr)
```
   string instr;
```

char *strindex(big,small)
```
   string big,small;
```

unsigned int stroctal(instr)
```
   string instr;
```

DESCRIPTION

A string is a pointer to a character vector. The actual definition is:

```
typedef char *string;
```

Strings are usually dynamically allocated, as opposed to char[]'s. It is uncool to use a string as a pointer into the middle of another string, but it can be done. (Use char * instead.)

Data in a string is followed by a NULL (or 0). This terminator must be included in the string length. You may create and initialize strings using:

```
 string foo = "This is the string value.";
 char bar[] = "Bar is also a string.";
 char baz[] = {'U','g','h','!',0}
```

The keyword "static" should be used if the declaration is global and the variable is not to be made available externally.

Stringlib routines:

fold(outstr,instr,whichway) -> string
folddown(outstr,instr) -> string
foldup(outstr,instr) -> string

Perform case folding. The input string is copied to the output string with the specified case folding. The same string may be specified as both "instr" and "outstr". The address of "outstr" is returned for convenience.

leftmatch(big,small) -> boolean

Return TRUE if initial characters of big match small exactly; else FALSE.

path(filename,direc,file)

Break a filename into directory and file strings. Direc and file are output buffers (user-supplied). File will not have any trailing /; but direc might.

The handling of most names is obvious, but several special cases exist. The name "f", containing no slashes, is split into directory "." and filename "f". Filename "/f" becomes "/" and "f". Filename "/" becomes "/" and ".". Filename "" becomes "." and ".". Trailing /s are ignored (except as the first character).

rootcomp(filename0,filename1) -> int

Compares the roots of two file names and returns the strcmp() value (0 for equality).

srchscore(big,small) -> int

Perform approximate string matching. Tells how well "small" matches substrings of "big". The score ranges from 0 to (strlen(small)**2). The score is something like the sum, over the longest matching contiguous substrings, of the square of the length of the substring minus some penalty for not-quite-exactly-matching substrings. Case is not significant.

strcopy(instr) -> string

Copy a string. This is one way to convert a substring or static string to a dynamic string.

strhex(instr) -> unsigned int

Convert ascii to hexadecimal. Strhex() converts the hexadecimal string to an unsigned integer. The string may contain leading blanks and tabs.

Conversion stops at the first character which is not a valid hexadecimal digit. The string of digits may optionally begin with "0x" or "0X", which will be ignored; the digits themselves include the characters "0" through "9", "a" through "f", and "A" through "F".

strindex(big,small) -> char *
: Find the address of one string within another. Strindex() searches for a substring of big which matches small, and returns a pointer to this substring. If no matching substring is found, 0 is returned. 0 is always returned if "small" is the null string.

stroctal(instr) -> unsigned int
: Convert ascii to octal. Stroctal() converts the octal string to an unsigned integer. The string may contain leading blanks and tabs. Conversion stops at the first character which is not a valid octal digit.

FILES

/iu/tb/include/stringlib.h
/iu/tb/lib/sublib/stringlib/*
/iu/tb/lib/sublib.a

SEE ALSO

atof(3), listlib(3), lstlib(3), parselib(3), strcmp(3)

DIAGNOSTICS

Some of the stringlib routines use the printerr error reporting system. An error will typically cause a message to be printed on the error stream and the routine will return a null string. Other routines use special return codes to flag errors.

BUGS

Beware of naming any variable "string", since the compiler will try to expand it as a type declaration. This applies to any routine which includes testbed.h directly or indirectly.

Rootcomp() requires string pointers instead of strings. The reason for this is not known.

Strhex() and stroctal() have no provision for overflow. Unary + is not accepted, and unary - is not accepted due to the interpretation of the numbers as unsigned.

HISTORY

03-Feb-83 Laws at SRI-IU
: Created this document.

21-Dec-81 Laws at SRI-IU
: Changed stlmatch() to leftmatch(), atoh() to strhex(), sindex to strindex, and atoo() to stroctal().

05-Dec-79 Steven Shafer (sas) at Carnegie-Mellon University
: Created atoh(), atoo(), the fold() routines, path(), stlmatch(), and sindex().

NAME

strlib – STR dynamic string manipulation package

SYNOPSIS

#include "strlib.h"

String descriptor (public fields only):

```
   typedef STR * struct {
      char *text;          Dynamically allocated text buffer.
        char *ptr;         Substring pointer.
        char *endptr;              End-of-buffer pointer.

        STR predecessor;   Pointer to a preceding STR;
        STR successor;             Pointer to a following STR;
   };
```

STR closestr(str)
   STR str;

STR copystr(instr,[outtxtlen])
   STR instr;

STR linkstr(instr)
   STR instr;

STR newstr(instring)
   string instring;

printstr(str)
   STR str;

boolean validstr(str)
   STR str;

DESCRIPTION

A STR is a header structure for a dynamically allocated text buffer. Alternatively, it may be regarded as a small in-core file. The header allows strings to be recognized when used in contexts that might also allow integers or other objects. It also allows manipulation of text buffers that may contain nulls, and provides an easy way to pass command string pointers of the kind used in parselib and (soon) in arglib.

Public fields are listed above. They should usually be treated as read-only except for the ptr, which corresponds to a file read/write pointer. References to the fields should have the form "str->ptr", where str is declared to be of type STR.

closestr(str) -> STR

If the STR is multiply linked, simply decrement the link count; otherwise free the STR structure and its associated buffer. The return value is generally NULL, but may be the original STR value if an error occurred.

copystr(instr,[outtxtlen]) -> STR
: Copy all fields of the STR to a new STR structure. The text buffer, including nulls, is also copied. The new buffer may be allocated to outtxtlen characters (plus a final null) if desired; the input text will be clipped or padded with nulls to fit.

linkstr(instr) -> STR
: Link to an existing string structure. Any change in one STR will also affect the other, but they may be closed independently. The predecessor and successor pointers are not affected.

newstr(instring) -> STR
: Create a new STR with the specified text. Currently there is no way to specify the allocated length; it will be just large enough to hold the string and its final NULL.

printstr(str)
: Print the elements of a STR structure, including up to 40 characters of the text buffer and the current substring. This is primarily used for debugging.

validstr(str) -> boolean
: Check for a non-null STR with a valid structure identification code.

FILES

/iu/tb/include/strlib.h
/iu/tb/lib/sublib/strlib/*
/iu/tb/lib/sublib.a

SEE ALSO

arglib(3), blklib(3), parselib(3), stringlib(3)

DIAGNOSTICS

The strlib routines use the printerr error reporting system. An error will typically cause a message to be printed on the error stream, and the routine will return a null STR.

HISTORY

09-Dec-83 Laws at SRI-IU
: Created.

NAME

system – issue a shell command

SYNOPSIS

system(string)
char *string;

DESCRIPTION

System causes the *string* to be given to *sh*(1) as input as if the string had been typed as a command at a terminal. The current process waits until the shell has completed, then returns the exit status of the shell.

FILES

/iu/tb/lib/sublib/syslib/system.c
/iu/tb/lib/sublib.a

SEE ALSO

popen(3), exec(2), wait(2)

DIAGNOSTICS

Exit status 127 indicates the shell couldn't be executed.

HISTORY

06-Dec-79 Steven Shafer (sas) at Carnegie-Mellon University
Modified for CMU – now performs path search for shell (see execlp(2)) instead of directly executing "/bin/sh".

NAME

timelib – time and date functions

SYNOPSIS

```
#include <sys/types.h>
#include <time.h>

char *cdate (timval);
  long *timval;

long gtime(buf);
  struct tm *buf;

pdate(datestr,tm)
  char *datestr;
  struct tm *tm;

time_t strtime(datestr)
  char *datestr;
```

DESCRIPTION

Cdate takes a long integer time value, pointed to by *timval*, and returns a pointer to a string representing the date. The string is of the form "dd-mmm-yy", where "dd" is the day number, "mmm" is an abbreviation for the month name, and "yy" is the last two digits of the year number.

The *time*(2) procedure is useful for determining the current date and time.

Cdate is very much like *ctime*(3), but returns a string representing only the date rather than the date and time.

Gtime reconstructs the long integer representation of the time from a buffer, *buf*, produced by the *localtime*(3) routine. In fact, *gtime* is the inverse of *localtime*.

Pdate parses a date string specification and fills in the appropriate fields of the supplied *tm structure* (only the first six fields are set).

A legal date is either a day, a time or a day followed by a time. Any date may also begin with a day of the week name (e.g. Sunday, Monday, etc.) which is parsed but ignored. For compatibility with the *ctime*(3) subroutine, dates of the form

```
<weekday> <monthname> <day> <year> <time>
```

are also accepted.

Days may be specified as follows:

```
<monthnum>/<day>/<year>
<day>-<monthname>-<year>
<day> <monthname> <year>
<monthname> <day> <year>
```

with the year optional in all cases. Times are specified as:

    <hours>:<minutes>:<seconds>

with the seconds field optional.

Months should be either spelled out or specified numerically as indicated above. Years may be specified in either two or four digit style. Commas, spaces, tabs and newlines may be used interchangeably to delimit fields according to individual taste. Both month names and days of the week may be uniquely abbreviated. Case is ignored for both.

EXAMPLE

The following are legal date specifications:

Tue Jan 1 11:56 1980
3-December-80,14:23:00
March 4, 1984 11:01
12/22/79
12:00

Strtime converts an ascii date specification to its equivalent UNIX date/time representation. The date string is parsed using *pdate*(3). The month, day and year default to today while the time defaults to 00:00:00.

FILES

/iu/tb/lib/sublib/timelib/*
/iu/tb/lib/sublib.a

SEE ALSO

time(2), localtime(3)

DIAGNOSTICS

Pdate returns 0 if date is parsed successfully. Returns -1 on error with the contents of the *tm* structure undefined.

Strtime returns the converted time, -1 for invalid dates.

BUGS

The length of the date string is limited to 50 characters.

HISTORY

21-Dec-81 Laws at SRI-IU
    Changed name from atot to strtime.

21-Feb-80 Mike Accetta (mja) at Carnegie-Mellon University
    Changed date string limit from 25 to 50 characters.

03-Jan-80 Mike Accetta (mja) at Carnegie-Mellon University
    Created atot.

06-Dec-79 Steven Shafer (sas) at Carnegie-Mellon University
    Created.

NAME

vectorlib – vector and memory manipulation routines

SYNOPSIS

blkmove(here,there,length)
char *here,*there;

cpyvct(inbgn,inknd,npos,outknd,outbgn)
pointer inbgn,outbgn;

nulvct(buffer,buflen)
char *buffer;

smear(addr,length,val)
char *addr;

char *stalloc(size)

DESCRIPTION

Blkmove() moves a block of *length* bytes from *here* to *there.*

Cpyvct() copies a vector and changes its storage mode. Inknd and outknd are integer (or enum) values describing the data pointed to by inbgn and outbgn. See datatype.h in /iu/tb/lib/sublib/vectorlib for the definitions. Npos is the number of data values to be copied. The input and output vectors may start at the same position; any other overlap may cause serious errors. It is the user's responsibility to guarantee that the pointers indicate vectors of adequate length and boundary alignment.

Nulvct() sets each byte of a buffer to binary 0.

Smear() sets each byte in a block of *length* bytes to *val*. The starting address of the block is given by *addr*.

Stalloc() allocates a block of at least *size* bytes, beginning on a word boundary, and returns a pointer to it. The space is allocated on the process's stack. (The allocation grain size is 4 bytes on the VAX.) The space will be reclaimed automatically when the procedure returns which called *stalloc()*.

FILES

/iu/tb/lib/sublib/vectorlib/*
/iu/tb/lib/sublib.a

DIAGNOSTICS

If *size* is not positive, or the stack would become bigger than the system allows, *stalloc* returns zero.

FILES

/iu/tb/lib/sublib/vectorlib/stalloc.c
/iu/tb/lib/sublib.a

SEE ALSO

malloc(3)

BUGS

Cpyvct() has not been tested in its current installation. The include file needs to be moved to /iu/tb/include and perhaps integrated with testbed.h. This is a fast version that uses simple assignment instead of subroutine calls to convert one data type to another. Clips and other range errors are not prevented or counted.

Nulvct() is probably slower than smear(). It could be speeded up by a factor of five using VAX assembly code.

The maximum stack size (currently 2^19 bytes, or 1/2 Megabyte) is wired into the stalloc() because there is no system call or #include file which could provide it.

Some of these routines may someday be rewritten to use a blklib-style VCT structure.

HISTORY

11-Feb-83 Laws at SRI-IU
Combined existing man pages. Added nulvct() and cpyvct().

28-Nov-80 David Smith (drs) at Carnegie-Mellon University
Created blkmove(), smear(), and stalloc().

NAME

wdwlib – routines for manipulating window structures

SYNOPSIS

#include "wdwlib.h"

Window descriptor (public fields only):

```
typedef WDW * struct {

    float minx;          Left bound.
    float miny;          Lower bound.
    float maxx;          Right bound.
    float maxy;          Upper bound.
    float xrange;        Maxx-minx.
    float yrange;        Maxy-miny.

    int mincol;          Greatest integer <= minx.
    int minrow;          Greatest integer <= miny.
    int maxcol;          Least integer >= maxx.
    int maxrow;          Least integer >= maxy.
    int ncols;           Maxcol+1-mincol.
    int nrows;           Maxrow+1-minrow.
};
```

WDW clipwdw(inwdw,limitwdw)
WDW inwdw,limitwdw;

WDW closewdw(wdw)
WDW wdw;

WDW copywdw(inwdw)
WDW inwdw;

WDW linkwdw(inwdw)
WDW inwdw;

WDW newintwdw(mincol,minrow,maxcol,maxrow)

WDW newpntwdw(minpnt,maxpnt)
PNT minpnt,maxpnt);

WDW newwdw(minx,miny,maxx,maxy)
float minx,miny,maxx,maxy;

printwdw(wdw)
WDW wdw;

boolean validwdw(wdw)
WDW wdw;

WDW wdwarg(cmdarg,limitwdw,dftwdw,[askmsg])

```
WDW limitwdw, dftwdw;
string askmsg;
```

DESCRIPTION

A WDW is an abstract grouping of coordinates to represent a two-dimensional region of an image, picture file, or array. Public fields are listed above: they should usually be treated as read-only. References to the fields should have the form "wdw->ncols", where wdw is declared to be of type WDW.

clipwdw(inwdw,limitwdw) -> WDW
: Return a new window which contains the inwdw coordinates clipped to the limitwdw ranges. (This essentially returns the window intersection.)

closewdw(wdw) -> WDW
: If the WDW is multiply linked, simply decrement the link count; otherwise free the WDW structure. The return value is generally NULL, but may be the original WDW value if an error occurred.

copywdw(inwdw) -> WDW
: Copy all fields of the window to a new WDW structure. The two WDWs will then be entirely separate.

linkwdw(inwdw) -> WDW
: Link to an existing window structure. Any change to one WDW will also affect the other, but they may be closed independently.

newintwdw(mincol,minrow,maxcol,maxrow) -> WDW
: Create a new WDW with the specified integer coordinates. The corresponding floating-point coordinates will be generated. The routine will accept the min and max coordinates in either order, but will print a warning message if it must reverse them.

newpntwdw(minpnt,maxpnt) -> WDW
: Create a new WDW with the specified corner points. The routine will accept the min and max points in either order, but will print a warning message if it must reverse them.

newwdw(minx,miny,maxx,maxy) -> WDW
: Create a new WDW with the specified floating-point coordinates. The enclosing integer coordinates will be generated. The routine will accept the min and max coordinates in either order, but will print a warning message if it must reverse them.

printwdw(wdw)
: Print partial contents of a WDW structure. This is primarily used for debugging.

validwdw(wdw) -> boolean
: Check for a non-null WDW with a valid structure identification code. This

does not currently check for consistency of the coordinate ranges.

wdwarg(cmdarg,limitwdw,dftwdw,[askmsg]) -> WDW
: Interactively obtain a WDW specification. Limitwdw may be NULL if there is no limit on the window size; dftwdw may be NULL if there is no default. Askmsg may be NULL, a string, or a LST of prompt strings (initial prompt, lower left point, and upper right point, additional help messages). See arglib(3) for further details.

FILES

/iu/tb/include/wdwlib.h
/iu/tb/lib/sublib/wdwlib/*
/iu/tb/lib/sublib.a

SEE ALSO

arglib(3), blklib(3), frmlib(3), imglib(3), pntlib(3), seglib(3)

DIAGNOSTICS

The wdwlib routines use the printerr error reporting system. An error will typically cause a message to be printed on the error stream and the routine will return a null WDW.

BUGS

Do not try to store infinities. FLOATINF is much larger than INF, so that compatibility is a problem. Further, a window defined from NEGINF to INF has a width too large to be represented by ncols. (There are currently no checks for such overflow problems.) It is usually cleaner to use a NULL window than an infinite one, although representing one that is infinite in only one dimension is a problem.

HISTORY

31-Jan-83 Laws at SRI-IU
: Added newpntwdw() and wdwarg() documentation.

16-Nov-82 Laws at SRI-IU
: Created this document.

NAME

windowlib – Grinnell graphics windowing system

SYNOPSIS

```
#include "window.h"
#include "dpytype.h"
#include "imgtype.h"


adraw(dpy, overlay, line_width, xpos, ypos);
   DPY *dpy;
   double xpos, ypos;

aerase(dpy, overlay, line_width, xpos, ypos);
   DPY *dpy;
   double xpos, ypos;

amove(dpy, xpos, ypos);
   DPY *dpy;
   double xpos, ypos;

clear(dpy, clear_flag);
   DPY *dpy;

cursor(dpy, xpos, ypos);
   DPY *dpy;
   double xpos, ypos;

DPY *dpyinit(display_mode);

fitpic(dpy, image_pointer);
   DPY *dpy;
   IMGDPY *image_pointer;

infodpy(dpy);
   DPY *dpy;

rdraw(dpy, overlay, line_width, xpos, ypos);
   DPY *dpy;
   double xpos, ypos;

rerase(dpy, overlay, line_width, xpos, ypos);
   DPY *dpy;
   double xpos, ypos;

rmove(dpy, xpos, ypos);
   DPY *dpy;
   double xpos, ypos;

setbox(dpy, box_mode);
   DPY *dpy;
```

setover(dpy, overlay_setting);
```
   DPY *dpy;
```

setsize(dpy, xstart, ystart, xlength, ylength);
```
   DPY *dpy;
   double xstart, ystart;
   double xlength, ylength;
```

setscale(dpy, xscale, yscale);
```
   DPY *dpy;
   double xscale, yscale;
```

DPY *show(dpy, image_pointer);
```
   DPY *dpy;
   IMGDPY *image_pointer;
```

DPY *showctr(dpy, image_pointer, xloc, yloc);
```
   DPY *dpy;
   IMGDPY *image_pointer;
```

text(dpy, overlay, size, direction, message);
```
   DPY *dpy;
   char *message;
```

textctr(dpy, overlay, text_string);
```
   DPY *dpy;
   char *text_string;
```

DPY *window(dpy, xstart, ystart, xlength, ylength);
```
   DPY *dpy;
   double xstart, ystart;
   double xlength, ylength;
```

w_setpixel(dpy, x, y, value);
```
   DPY *dpy;
   double x, y;
```

w_setcpixel(dpy, x, y, red_value, green_value, blue_value);
```
   DPY *dpy;
   double x,y;
```

w_getpixel(dpy, x, y, value);
```
   DPY *dpy;
   double x, y;
```

w_getcpixel(dpy, x, y, red_value, green_value, blue_value);
```
   DPY *dpy;
   double x, y;
```

w_setblkpix(dpy, xstart, ystart, xend, yend, value);
```
   DPY *dpy;
   double xstart, ystart, xend, yend;
```

w_setcblkpix(dpy, xstart, ystart, xend, yend, rv, gv, bv);
DPY *dpy;
double xstart, ystart, xend, yend;

DPY *w_rdimg(dpy, image_pointer, xstart, ystart, xend, yend, x, y);
DPY *dpy;
IMGDPY *image_pointer;
double x, y;

DESCRIPTION

This Grinnell windowing system, based on the CMU Grinnell subroutine library, allows the actuall Grinnell screen to be divided into a number of logical windows. Each window is described by a dpy data structure. The subroutines in this package allow the user to treat each window as a small Grinnell, and not to be concerned with the location of the window on the actual Grinnell.

Each window has its own coordinate system which initially is set from 0 to 1. When an image is shown in the window a new dpy is created with a coordinate system that matches that of the image. In addition it is possible to set the window coordinate system to any desired values.

Routines exist to draw lines in the Grinnell overlays, using the window coordinate system, to show pictures, to use the cursor, and to do various other things.

SUBROUTINES

adraw(dpy, overlay, line_width, xpos, ypos);
Adraw will draw a line from the current position to the absolute position (xpos, ypos) in the window.

aerase(dpy, overlay, line_width, xpos, ypos);
Aerase will erase a line from the current position to the absolute postion (xpos, ypos) in the window.

amove(dpy, xpos, ypos);
Set the current position in the window to (xpos, ypos).

clear(dpy, clear_flag);
Clear the window specified by dpy. The bits set in clear_flag indicate which overlays should be cleared. Bits 0-3 are for the four overlays and bit 4 clears the display.

cursor(dpy, xpos, ypos);
Return the position of the cursor in the window specified by the dpy. The position (xpos, ypos) is in the window coordinate system.

DPY *dpyinit(display_mode);
Intialize the Grinnell and the window system. The display_mode sets the Grinnell up for (0) black and white images or (1) RGB color images. Dpyinit returns a pointer to a dpy representing the entire Grinnell screen, with a unit window coordinate system.

fitpic(dpy, image_pointer);
Fitpic will set the pixel scale factor in the dpy so the entire image can be displayed.

infodpy(dpy);
Infodpy will print out the contents of a dpy on the standard output device.

rdraw(dpy, overlay, line_width, xpos, ypos);
Rdraw will draw a line from the current location to the relative location offset be (xpos, ypos). The line will be drawn in the overlay specified by overlay, in the size specified by line_width.

rerase(dpy, overlay, line_width, xpos, ypos);
Rerase will erase a line in the specified overlay from the current location to the relative location offset by (xpos, ypos).

rmove(dpy, xpos, ypos);
Set the current location to the location specified by the offset (xpos, ypos) from the current location.

setbox(dpy, box_mode);
The setbox command sets the box field in the specified dpy. When box mode is in effect a boader, or box, is drawn around the window on the Grinnell. Box_mode is an overlay value that specifies which overlay the box is to be drawn in. For the box to be shown on the Grinnell that overlay must be turned on (see the setover function).

setover(dpy, overlay_setting);
Setover sets the active overlay flags in a dpy. The overlay flags control which overlays are turned on when an image is shown in a dpy.

setsize(dpy, xstart, ystart, xlength, ylength);
Setsize sets the window coordinate system in a dpy. When a window is intially made is has a coordinate system from 0 to 1. Setsize can be used to change the coordinate system. Note: When an image is displayed the coordinate system of the dpy is adjusted to match the coordinate system of the image.

setscale(dpy, xscale, yscale);
Setscale changes the pixel scale factors in a dpy. The scale factor times the number of pixels in an image represents the number of pixels that image will have when displayed in the window.

DPY *show(dpy, image_pointer);
The show command is used to show pictures in a window. The image is centered in the window. If the image is larger than the window it will be truncated. Show returns a pointer to a new dpy that has a coordinate system to match that of the image shown in the window. If show is given a null image/pointer the Grinnell will be updated with the overlay and box

settings in the dpy.

DPY *showctr(dpy, image_pointer, xloc, yloc);
Showctr is similar to show, except the loction in the image (xloc, yloc) is forced to be in the center of the window. Any part of the image that does not fit in the window is truncated.

text(dpy, overlay, size, direction, message);
Text is used to write the text information, contained in message, on the Grinnell. Two letter sizes are supported, small (1) and large (2). The direction controls the direction that the text is displayed in. Three directions are possible, horizontal (1) vertical (2) and diagonal (3).

textctr(dpy, overlay, text_string);
Textctr will center the text at the current location. As usual, overlay specifies the overlay that the text will be drawn in. It is only possible to draw text in the horizontal direction.

DPY *window(dpy, xstart, ystart, xlength, ylength);
Window is used to cut a window in a dpy. (xstart, ystart) specifies the starting location of the new window in the dpy. (xlength, ylength) specifies the length of the window. The coordinate system of the new window is initialy set to unity. The routines returns a pointer to a dpy that describes the new window.

w_setpixel(dpy, x, y, value);
W_setpixel will set pixel (x,y) in the window described by the dpy to the specified value.

w_setcpixel(dpy, x, y, red_value, green_value, blue_value);
W_setxpixel will set the color pixel in the dpy at location (x,y) to the color value specified.

w_getpixel(dpy, x, y, value);
W_getpixel will get the pixel (x,y) from the window and assign its value to value.

w_getcpixel(dpy, x, y, red_value, green_value, blue_value);
W_getcpixel will get the color pixel from window dpy at location (x,y), and assign its value to the specified variables.

w_setblkpix(dpy, xstart, ystart, xend, yend, value);
W_setblkpix will set the specified block of pixels to the given value.

w_setcblkpix(dpy, xstart, ystart, xend, yend, rv, gv, bv);
W_setcblkpix will set the specified block to the given color value.

DPY *w_rdimg(dpy, image_pointer, xstart, ystart, xend, yend, x, y);

W_rdimg will read an image specified by image_pointer into the Grinnell. The image file, bounded by (xstart, ystart) and (xend, yend) will be put in the window starting at location (x,y). Any part of the image that does not fit in the window will be truncated. If the pixel scale factor is less then 1, the image will be reduced as needed. The routine returns a dpy with a coordinate system that matches that of the displayed portion of the image.

FILES

/iu/tb/include/window.h
/iu/tb/include/dpytype.h
/iu/tb/include/imgtype.h
/iu/tb/lib/visionlib/windowlib/*
/iu/tb/lib/visionlib.a

SEE ALSO

dpy(1), dsplib(3), frmlib(3), wdwlib(3)

BUGS

Many of these routines should be rewritten to use testbed imglib and wdwlib routines, and perhaps also frmlib routines. The DPY structure should be rewritten in blklib form, and might be developed into a PORT or VIEW structure.

HISTORY

30-Sep-83 Laws at SRI-IU
Converted from CMU IMAGE form to testbed PIC form.

05-Aug-82 Siegel at SRI-IU
Created.

PART 5 — File Formats

NAME

camdist – file formats used in camera calibration

DESCRIPTION

This manual entry contains descriptions of the formats of the files used by the program *camdist*. All files are ASCII, and are opened with 'fopen', read with 'fscanf', and written with 'fprintf'. Variable names listed below are those used in *camdist*. Variables are read assuming at least one space between successive values. The following descriptions organize the data onto separate lines of a file. This is for convenience only; the data could just as well be merged onto a single line.

The **matched point files** contain the film plane data which is the primary input to *camdist*, and should contain:

```
NPOINTS
X1[1] Y1[1] X2[1] Y2[1] SXX[1] SYY[1] SXY[1]
X1[2] Y1[2] X2[2] Y2[2] SXX[2] SYY[2] SXY[2]
X1[3] Y1[3] X2[3] Y2[3] SXX[3] SYY[3] SXY[3]
...
X1[n] Y1[n] X2[n] Y2[n] SXX[n] SYY[n] SXY[n]
```

where the variables are:

int NPOINTS	The number of point pairs in the file.
float X1,Y1	The co-ordinates of the point in camera 1's film plane.
float X2,Y2	The co-ordinates of the point in camera 2's film plane. Co-ordinates are given with respect to the center of the image, with X increasing to the right, Y increasing upward.
float SXX,SYY	The variances (square of standard deviations) of expected errors of X2 and Y2 in the film plane. (These numbers affect the accuracy of the solution--making them too big will cause the errors in the solution to be huge. However, because the code uses these numbers to normalize various intermediate calculations, they cannot be set to zero, either. The code enforces a minimum of 10^-10 on these values.)
float SXY	The covariance of X2 vs Y2 for a given point. (The covariance of the errors between different points is assumed to be zero.)

No attempt is made to do error recovery if there are fewer than NPOINTS data points in the file.

The **camera model file** format is used both for input of suggested initial values for starting *camdist* and for output of the solution found. The values expected in the file are:

```
FOCUS1   FOCUS2
GPR[1]   SPR[1]
GPR[2]   SPR[2]
GPR[3]   SPR[3]
GPR[4]   SPR[4]
GPR[5]   SPR[5]
BASEDIST SIGR
```

where the variables are:

float FOCUS1	The effective focal length of camera 1.
float FOCUS2	The effective focal length of camera 2. Focal lengths must be expressed in the same unit as the image plane distances, usually pixels.

float GPR The angles which describe the relative positions and orientations of the two cameras. The angles and the conventions on their signs are:

1 Azimuth from camera 1 to camera 2, clockwise as seen from above.

2 Elevation from camera 1 to camera 2, upward.

3 Pan of camera 2, clockwise as seen from above.

4 Tilt of camera 2, upward.

5 Roll of camera 2, clockwise looking out through the camera.

On input and output, these angles are in degrees. The initial estimate of these angles should be reasonably close to the solution; anything more than 90 degrees away will probably result in the wrong solution being found.

float SPR	The standard deviations of the angles in GPR. On input to the camera solver, these numbers should be quite large (1000 is reasonable), as they are used to determine how closely the solution should adhere to the given initialization. On output, they are the estimated error in the various angles.
float BASEDIST	The known distance between the camera lens centers, used for the distance calculations. If this is unknown, use 1.0; the output distances will then be in multiples of the baseline distance.
float SIGR	The standard deviation of the error in the baseline distance. (Zero is reasonable.)

No attempt is made to ensure that the file contains enough data.

A **distance file** is the output of the distance calculation portion of *camdist*, and contains:

```
NPOINTS
X1[1] Y1[1] Z[1] SIGZREL[1]
X1[2] Y1[2] Z[2] SIGZREL[2]
X1[3] Y1[3] Z[3] SIGZREL[3]
. . .
X1[n] Y1[n] Z[n] SIGZREL[n]
```

where the variables are:

int NPOINTS	The number of points in the file. This may be less than the number of points in the original match file, since no distance is computed for points which have been edited out.
float X1,Y1	The point's film plane position in the first camera.
float Z	The z-component of the points' 3-space positions, that is, the

	distance from the focal point to the plane (parallel to the film plane) containing the 3-space point.
float SIGZREL	The computed standard deviations of the errors in Z, representing an upper limit to the relative error in the points.

The **parameter file** is used for input of the optional error propagation parameters for *camdist*, and should contain:

```
SDP SDF CORDST MAXEDIT
SIGRAN  SIGSYS  TOL
```

where the variables are:

float SDP	The additional standard deviation of the observation errors. (Default value 0.01)
int SDF	Weight to be given SDP^2 in the variance adjustment. (SDP^2 plus the mean observation error propagated from SXX, SYY, and SXY is considered to have a Chi-square distribution with SDF degrees of freedom.) The default=0 means that the variance is completely free to be adjusted according to the residuals.
float CORDST	The correlation distance for SDP. The errors represented by SDP are considered to have a correlation equal to *exp*(-D^2/(2*CORDST^2)), where D is the distance between the pair of points in the film plane. An approximate correction for this correlation is made in the adjustment. A positive value of CORDST may slow the program down considerably if NPOINTS is large. If CORDST is negative, a cruder but faster approximation is used which assumes that the points are uniformly distributed in the film plane between their actual minimum and maximum x- and y-values. If CORDST=0 (the default), the correlation of different points is always zero, thus the entire distancing computation is avoided.
int MAXEDIT	The maximum number of points to remove by editing (default 5). The purpose of this control is to limit wasted computer time in case of a runaway bad solution. If MAXEDIT=0, no editing will be done, but the processing will abort because the procedure usually tries to remove one more point than it finally decides upon.
float SIGRAN	Standard deviation of random errors in the X2 and Y2 measurements, in addition to that represented by SXX, SYY, and SXY. (Default=0.0)
float SIGSYS	Standard deviation of systematic errors in the X2 and Y2 measurements, correlated between different points. The default is to set SIGSYS to -1, which causes it to be set to *sqrt*(SD^2-SIGRAN^2), where SD is the standard deviation obtained from the camera calibration.
float TOL	The tolerance for computing complete error propagation. If TOL is zero (the default), the complete error propagation is done for all points. Otherwise, computations for the error in Z use information computed for nearby points in order to save time, with increasing loss of accuracy as TOL increases. If TOL is small compared to 1, the results are fairly accurate. If TOL is considerably

greater than 1, the maximum saving of time will be achieved.

SEE ALSO

camdist(1)

HISTORY

18-Mar-83 Hannah at SRI-IU
Created

NAME

docformat – Testbed documentation record format

DESCRIPTION

An image file may include several types of descriptor records. The picture file itself may be described by a FILEHDR record. Information about the image data format is contained in an IMGHDR record. The image data itself is then stored using the format described by the IMGHDR. Free-form comments about data collection parameters, image transformations, or physical significance of the data are permitted in an IMGDOC trailer following the image data.

A picture file may have one IMGDOC record, the last record in the file. It consists of a few descriptor fields followed by any number of free-format descriptors separated by newline characters. The initial descriptive fields are fixed-format to simpify machine reading of the image format. Each field is eight characters long. (A few of the fields are composed of smaller fixed-size units.) Character fields should be left-adjusted (i.e., padded with blanks on the trailing end). Numeric fields should be right-adjusted and padded with blanks. A zero field must be indicated by a zero digit.

Descriptors are meant to record the history and current status of the image data. New descriptors are written after old ones. A site may choose to update the descriptor fields (e.g., MEAN-VALUE or STANDARD-DEVIATION) whenever the image data is changed, or it may leave the old descriptors and append the new values at some later point. A site may also indicate transformations without either deleting old descriptors or appending new ones: let the user beware. (It is very bad form, however, to leave obsolete descriptors in the trailer without indicating that a transformation has been done.)

The optional IMGDOC trailer may contain any type of information. The documentation is usually a text field following the IMGDOC record descriptor fields. This text field could contain additional keywords (e.g., FREEFORM or PROPLIST), but no such system has yet been established.

Human-readable ASCII data is generally preferred, although NULLs should be used instead of blanks if it is neccesary to pad the IMGDOC record to the end of a file (e.g., to overwrite an old descriptor. Standard IMGDOC parsing software will return just the text up to the first NULL.

```
#define IMGDOCLEN   24              Length of IMGDOC ID fields.


struct {

  char formattype[8];              Format, version: "SRI TB01".
  char recordtype[8];              Header/trailer type: "IMGDOC  ".
  char recordbytes[8];             Length of this record.   (>= IMGDOCLEN)

  char text;                       Text string to NULL or end of file.

} IMGDOC;
```

Restricted IMGDOC Grammar

The following is a restricted grammar that can be used for maximum clarity and minimum problems during image transmission. The grammar is presented only as an example, and no endorsement is implied.

The full ASCII character set may be used in trailer records. (For maximum intersite compatibility this could be limited to the BCD character subset -- letters A-Z (octal 101-132), digits 0-9 (060-071), special symbols "$ ' ( ) * + , - . / =" (044, 047-057, 075), blank (040), and newline (015). The restriction to a BCD-compatable character set is not necessarily optimal. It attempts to promote hypothetical data interchange by restricting the content of the documentation. This makes documentation of an image more difficult, and may thus result in poorer documentation.)

Each descriptor must have the form "name = value". A comment descriptor is one that does not conform to this syntax. Blanks serve only to delimit tokens, and may be used freely. (Blanks should also be used as a fill character to pad to the end of the IMGDOC record.) Descriptors are terminated by a newline that is not hidden by parentheses. End of file marks the end of the trailer data.

A descriptive name is a character string that begins with a letter and contains only letters, digits, and the special symbols "* + - . /". A value may be a number, string, name, or LISP expression. Each value may be followed by the name of the units in which it is measured. (Default units may be assumed if none are specified.) The precise syntax is shown below.

Example:

```
THIS IS AN INTRODUCTORY COMMENT.  IT CAN BE
 SPREAD OVER SEVERAL LINES.

IMAGE-NAME   = 'SAN FRANCISCO BAY'
FOCAL-LENGTH = 12.0 INCHES
IMAGE-DATE   = (DAY = 7, MONTH = 3, YEAR = 1978)

PHOTOMETRIC-TRANSFORMATION = (PLUS (TIMES I 3.489) 1000.8) LUMENS/SQ.FT
```

Formal Syntax

In the following grammar "|" delimits alternative constructs; "{" and "}" bound special constructs. Spaces and newlines may be used freely anywhere except within a name, integer, or real; these tokens are thus ended by a terminator. The symbol NIL is ambiguously defined: it may be considered a name or a literal. A comment is any sequence of characters that cannot be parsed as a descriptor. The trailer is a sequence of descriptors or comments plus the necessary fixed-format IMGDOC record descriptors.

```
 proplist          :=  descriptor | proplist breakchar descriptor
```

```
descriptor     := name = value | comment
comment        := {sequence of characters between newlines}

value          := val | val unit | ( value ) | ( proplist )
val            := atom | s-exp
unit           := name

s-exp          := NIL | atom | ( s-explist )
s-explist      := s-exp | s-explist s-exp
atom           := name | integer | real | string

integer        := sign number | number
number         := digit | number digit

real           := sign floatnumber | floatnumber
floatnumber    := mantissa | mantissa E integer
mantissa       := integer | integer . | . number
                    | integer . number

name           := alpha | name alphanumeric
string         := ' {sequence of non-quote characters} '
                    | string string

alphanumeric   := alpha | digit | sign | * | . | /
alpha          := A | B | ... | Z
digit          := 0 | 1 | ... | 9
sign           := + | -
terminator     := space | newline | $ | , | = | ' | ( | )
breakchar      := , | newline
```

Suggested IMGDOC Parameters

Image transformations will usually be documented by giving the name and parameters of the transformation routine. This is convenient for retracing the history of an image, but nearly useless for information interchange between sites.

Information can best be captured and transmitted by constrained free-form descriptor fields of the type shown below. The following list is not exhausted, nor is it mandatory. Note that the descriptive fields should be strings, not comments.

Source Descriptors:

IMAGE-SOURCE	Originating institution.
IMAGE-DATE	Imaging date.
IMAGE-TIME	Local time or GMT.
IMAGE-NAME	Unique identifying name.

Scene Descriptors:

SCENE-NAME	Informal title.
SCENE-FEATURES	List of principal features.
SCENE-CONDITIONS	Atmospheric degradations.
SCENE-DESCRIPTION	Informal description.

Sensor Parameters:

SENSOR-TYPE	Physical sensor identifier.
SENSOR-CHARACTERISTICS	Focal length, etc.
PLATFORM-LOCATION	UTM grid or latitude/longitude/altitude.
VIEWING-DIRECTION	Heading, pitch, and roll.
PRINCIPAL-LOCATION	Ground location of the principal point.
SENSOR-FUNCTON	Positive/Negative/Response function.
SENSOR-BAND	Available subset of sensor output.

Digitization Parameters

DIGITIZATION-SITE	Facility doing the digitization.
DIGITIZATION-DATE	Date of processing.
DIGITIZATION-TIME	Local time or GMT.
SCANNER-TYPE	Physical hardware designation.
SCANNER-RESOLUTION	Resolution in microns, etc.
SCANNER-RESPONSE	Transmission/Density/Reflectance.
SCANNER-FUNCTION	Photometric response function.
DIGITIZATION-CENTER	Pixel address of the principal point.
DIGITIZATION-ORIENTATION	Scan direction relative to image axes.
QUANTIZATION-FUNCTION	Pixel-value intervals.

Transformation History

SOURCE-IMAGE	Parent image identifier.
SOURCE-ADDRESS	Principal point or window in source.
GEOMETRIC-TRANSFORMATION	Parameters of warp routine.
PHOTOMETRIC-TRANSFORMATION	Space-invariant pixel value transform.
FREQUENCY-TRANSFORMATION	Fourier or related transform.
FILTER-FUNCTION	Parameters of spatial filtering.

Current Status

PIXEL-TYPE	Scalar, complex, multiband, etc.
COLOR-CODE	RGB/YIQ/IHS or other color code.

Statistical Descriptors

MIN-VALUE	Lowest pixel value.
MAX-VALUE	Highest pixel value.
MEAN-VALUE	Arithmetic average pixel value.

STANDARD-DEVIATION	Root mean squared deviation.
HISTOGRAM	Vector of pixel value counts.

FILES

/iu/tb/include/tbdoc.h

SEE ALSO

doclib(3), imgformat(5), hdrformat(5)

BUGS

Use of image trailers for updatable documentation is somewhat dangerous since it requires that users write data to a variable location in the file. It is very easy to overwrite the image data in such circumstances.

The IMGDOC name is something of an anachronism. Doclib is now completely separate from the image access code, and the documentation records need have nothing to do with image documentation.

HISTORY

07-Feb-83 Laws at SRI-IU
Created.

NAME

hdrformat – Testbed image header format

DESCRIPTION

This file contains all of the information needed to reconstruct a two-dimensional image from a one-dimensional picture file in the SRI Image Understanding Testbed picture format. The current version of this format is known as SRI TB01.

The header fields are fixed-format to simpify machine reading of the image format. Each field is eight characters long. (A few of the fields are composed of smaller fixed-size units.) Character fields should be left-adjusted (i.e., padded with blanks on the trailing end). Numeric fields should be right-adjusted and padded with blanks. A zero field must be indicated by a zero digit.

The SRI TB01 data format is designed for ease of sequential processing without sacrificing ease of random-access processing. Critical data descriptors are placed ahead of the described data to facilitate sequential processing. Orthogonal pixel layout is preserved to permit pixel address computation using row and column offset vectors. Padding is permitted so that data block size can be matched to computer memory page size or data transfer buffer size.

Image data are stored in a block raster format consisting of rectangular data blocks that cover the image in a raster scan pattern. (Scan direction is recorded in the header scantype field.) Pixels within a data block are ordered in the same raster pattern. All data blocks in a picture file are the same size, and each must contain an integral number of pixels on each row. Any additional bits in the block row will be padded with zeros following the valid image data.

The array of valid data pixels is an image. The data is addressed by pixel within scan line and by scan line within the image. These may be thought of as rows and columns, but the relation to any physical scene depends on the scanning direction.

Data blocks are groupings superimposed on the linear stream of pixels and padding that form the data file. The basic unit is the block row, which contains an integral number of pixels and possibly some padding bits. Block rows are grouped into blocks, which contain rectangular portions of the image. Blocks are grouped into block strips, which contain an integral number of scan lines. The juxtaposition of these block strips forms the data file.

Currently defined image scan patterns are listed below. Scan directions are specified looking toward the original scene. The direction between adjacent image pixels is specified first, then the scan-to-scan direction. LRBTSCAN is customary for testbed image data; LRTBSCAN is the normal television raster (without interlace).

Lower-left origin:

LRBTSCAN:	pixels left to right, scans bottom to top.
BTLRSCAN:	pixels bottom to top, scans left to right.

Upper-left origin:

LRTBSCAN:	pixels left to right, scans top to bottom.

TBLRSCAN: pixels top to bottom, scans left to right.

Lower-right origin:
RLBTSCAN: pixels right to left, scans bottom to top.
BTRLSCAN: pixels bottom to top, scans right to left.

Upper-left origin:
RLTBSCAN: pixels right to left, scans top to bottom.
TBRLSCAN: pixels top to bottom, scans right to left.

Additonal scan and interlace patterns will be defined as needed.

The block raster format is designed for random access on a machine with virtual memory capability. Blocked structure is not mandatory, however. Each row (or column) of the image may be declared a separate block, in which case the image is simply raster scanned. Blocks may also be subrows if this is more convenient. No particular relationship is needed between data block size and image size.

For data transmission no relationship is needed between header or data block size and the record size of the physical storage medium. For random data access, however, it is usually desirable for the IMGHDR header and the data blocks to correspond to virtual memory pages. This will prevent pixels from being split across memory pages.

The array of data blocks may cover a slightly larger area than the image data, in which case the blocks will be padded with zeros at the trailing end of each scan direction. Every data block will contain at least one pixel. Degenerate images are permitted; one with no rows or columns will also have no data blocks. One with no data bands will have blocks consisting of padded pixels. Bands and pixels of zero bits are also permitted, although they may not be supported on all systems.

Pixel Formats

An image is assumed to be a rectangular array of pixels, with each pixel consisting of zero or more data bands. Pixels must all have the same format, but the format is arbitrary and depends only on the image access software.

Pixels may be composed of multiple elements called bands. Each band may be composed of arbitrary subfields (e.g., exponent and mantissa). General-purpose image access software will break out only the pixels; any access to the subpixel band elements is up to the user. Special- purpose access software may exist for particular pixel formats.

The following bandtype codes are currently defined, but not necessarily supported:

UNTYPED : arbitrary byte pattern.

UNSIGNED: unsigned binary integer.

TWOSCOMP: two's complement signed integer.

VAXREAL : VAX floating-point format.

VAXDBL : VAX double-precision format.

Complex pixels consisting of real and imaginary components may be treated as a primitive band element type or as a pair of image bands. (If they are treated as primitive, a bandtype code will be needed for each underlying numeric type, such as INTCPLX, REALCPLX).

Ordinarily the number of bits per element will be a power of two, and a multiple of the byte size if the element is at least one byte long. Band elements are typically unsigned integers, but can be of any type. Bands follow one another with no separating padding.

Each pixel must be large enough to hold its data bands; any unused bits at the end will be filled with zeros. There is no padding between pixels except at the ends of the data block rows.

It is suggested that most multiband images be stored with a separate picture file for each band. This facilitates single-band processing without degrading multiband processing on systems with disk storage and virtual memory. It also permits easy insertion of additional image bands (e.g., ground truth, texture energy planes, or classification maps). Manipulation of such a multiband image is simplified if all picture and descriptor files are stored in a separate directory.

IMGHDR Definition

Standard header record length. Other lengths may be used; see below.

```
#define IMGHDRLEN  1024            IMGHDR length.

typedef struct IMGHDR {

 char formattype[8];               Format, version "SRI TB01."
 char recordtype[8];               Header/trailer type "IMGHDR."
 char recordbytes[8];              Length of this record, IMGHDRLEN.

 char scantype[8];                 Scan direction; see text.

 char bandtype[8];                 Pixel band format code; see text.
 char imagebands[8];               Bands per pixel.
 char bandbits[8];                 Bits per band.
 char pixelbits[8];                Bits per pixel.

 char linepixels[8];               Pixels per scan line.
 char imagelines[8];               Number of image scan lines.
```

```
    char blockcols[8];                  Bytes per block row.
    char blockrows[8];                  Rows (scan lines) per block.

    char stripblocks[8];                Blocks per scan line strip.
    char imagestrips[8];                Strips per image.

    char unused[IMGHDRLEN-112];         Pad to recordbytes bytes.

} IMGHDR;
```

Potential Modifications

The IMGHDR is bit-oriented, although most access methods are byte oriented. The band and pixel data could be restricted to particular byte alignments, e.g. multiples of eight bits for unpacked data and powers of two for fields packed within a byte.

The stripblocks and imagestrips fields are redundant, and could be eliminated. They have been retained as a convenience and as an error check. They may become important if other padding schemes are permitted, or if images are allowed to fill in additional rows and columns up to some preformatted maximum.

There could be two alternate methods of packing image data into blocks. The sequential scan of image data could include padding along scan lines but no padding of missing scans, or it could omit scan padding altogether. Either method permits complete recovery of the image because the size and image content of each block can be calculated. These methods conserve storage (for odd image sizes) at the expense of somewhat more complicated pixel addressing.

Packing formats could be defined for compression of redundant data. Color transformations are often used for multiband images. Run length coding and predictive coding are useful for slowly varying imagery. Symmetric images (e.g., Fourier-domain data) can be reconstructed from half or one quarter of the data. Quad trees are useful for segmentation maps and for pyramid coding of images to make successive resolutions available. Block floating- point, in which a single normalizing exponent is used for an entire block of mantissas, is another packing scheme. At present such packing formats are unsupported. The current standard simply stores two-dimensional (or multidimensional) data; any repacking or interpretation is the user's responsibility.

The current IMGHDR is not very supportive of any data formats except raster-scanned and block-scanned storage. This is because such formats require descriptor fields other than those currently defined. While such fields can be added to the IMGHDR, this leaves the standard block-scan descriptors unused and "dangling". At present the best solution would be to define a different IMGHDR type for each type of image to be accessed. A more general solution, the use of free-form descriptors as in the IMGDOC record, is believed to be too complex given the current difficulty in porting images with even simple fixed-format headers.

Although not currently defined, there is nothing preventing the inclusion of access vectors in an IMGHDR record. These would be tables of pixel column and scan line offsets that can be added together to find the address of a given pixel within the image. Such tables would simplify the accessing of pyramid images, folded (symmetric) images, and certain other regular storage schemes. At present it is felt that these tables are easily built by the picture accessing software, and thus need not be stored as part of every image.

FILES

/iu/tb/include/tbhdr.h

SEE ALSO

hdrlib(3), imgformat(5), docformat(5)

BUGS

The "standard" header length of 1024 bytes may cause problems on some machines. A DEC 2060 prefers 2048 bytes.

HISTORY

07-Feb-83 Laws at SRI-IU
Created.

NAME

imgformat – Testbed image format

DESCRIPTION

SRI Image Understanding Testbed Picture Format

This file contains information needed to reconstruct a two-dimensional image from a one-dimensional file in the SRI Image Understanding Testbed picture format. Multiband images are usually stored as separate single-band picture files, although the format allows for multiband pixels within a file.

The format is intented for both picture processing use and ease of intersite transmission. Very few constraints have been placed on the storage format, although the full flexibility may not be supported at any given installation. The format was developed at SRI as part of the IU Testbed.

The format includes several types of descriptor records. The picture file itself may be described by a FILEHDR record. Information about the image data format is contained in an IMGHDR record. The image data itself is then stored using the format described by the IMGHDR. Free-form comments about data collection parameters, image transformations, or physical significance of the data are permitted in an IMGDOC trailer following the image data.

Header fields in uppercase ASCII are preferred to facilitate machine interpretation and interfacility transmission. When possible the character set should be restricted to the ASCII letters A-Z (octal 101-132), digits 0-9 (060-071), special symbols "$ ' ( ) * + , - . / =" (044, 047-057, 075), blank (040), and newline (015). This character set has been chosen for compatibility with the NATO image transmission standard and with the BCD character set commonly used on 7-track tapes.

The header fields are fixed-format to simpify machine reading of the image format. Each field is eight characters long. (A few of the fields are composed of smaller fixed-size units.) Character fields should be left-adjusted (i.e., padded with blanks on the trailing end). Numeric fields should be right-adjusted and padded with blanks. A zero field must be indicated by a zero digit.

FILEHDR Format

The optional FILEHDR record is intended only for transmission of image data between sites using incompatible picture formats. Although the record type itself is defined, its contents are not yet established. See below for a discussion of intersite transmission considerations.

The basic element of storage is the byte. A byte is usually eight bits, although other sizes are possible. The FILEHDR record may be used to specify that "logical" bytes of a given size are being stored within "physical" bytes of larger size; the type of padding used must also be indicated. Bytes may not be smaller than eight bits.

The currently defined fields are:

```
#define FILEHDRLEN  256                FILEHDR length.

struct {

  char formattype[8];                  Format, version: "SRI TB01".
  char recordtype[8];                  Header/trailer type: "FILEHDR ".
  char recordbytes[8];                 Length of this record (FILEHDRLEN).

  char lab[8];                         Source country, lab: e.g. "USA SRI ".
  char picdate[8];                     Picture creation date: e.g. "03MAY82 ".
  char picname[8];                     Arbitrary 8-character picture title.

  char bytebits[8];                    Physical bits per byte.
  char databits[8];                    Logical bits per byte.

  char headerbytes[8];                 Size of the IMGHDR record.
  char imagebytes[8];                  Size of the image data and padding.
  char trailerbytes[8];                Size of the IMGDOC trailer.

  char unused[FILEHDRLEN-88];          Pad to recordbytes bytes.

} FILEHDR;
```

IMGHDR Format

An IMGHDR structure is concerned only with the physical representation of image data. (The physical significance of the data itself should be spelled out in the IMGDOC trailer record following the image data. Often the pixels will be unsigned 8-bit integers representing image luminance or density values from 0 to 255.)

Information in the header must be sufficient to unambiguously reconstruct the image from the picture data. Nonstandard pixel formats should be documented in the following IMGDOC trailer. If not otherwise specified, it is assumed that integer pixels are coded in binary with the sign and most significant bit coming before the least significant bit, and that data is stored in eight-bit bytes.

More than one IMGHDR and image data format may be defined, although only one can be used in a single picture file. At present only the SRI TB01 format is defined. It is described below.

An IMGHDR record may include unassigned fields used as padding. It is poor form to use these fields for any other purpose without formally defining a new IMGHDR format. (Such use often conflicts with later attempts to generalize the format in an upward-compatible fashion.) Any extensions to the SRI TB01 format will be assigned a new version number.

See hdrformat(5) for further details.

IMGDOC Format

A picture file may have one IMGDOC record, the last record in the file. It consists of a few descriptor fields followed by any number of free-format comments. The comments will typically be separated by newline characters.

Descriptors are meant to record the history and current status of the image data. New descriptors are written after old ones. A site may choose to update the descriptor fields (e.g., MEAN-VALUE or STANDARD-DEVIATION) whenever the image data is changed, or it may leave the old descriptors and append the new values at some later point. A site may also indicate transformations without either deleting old descriptors or appending new ones: let the user beware. (It is very bad form, however, to leave obsolete descriptors in the trailer without indicating that a transformation has been done.)

See docformat(5) for further details.

Intersite Transmission

This picture file format is designed for unambiguous documentation of image data during transmission or storage. Although suitable for intersite picture transmission, the format does not explicitly consider tape headers, blocking factors, and other transmission parameters.

For intersite transmission three levels of conversion are forseen:

- Transmit the data in its current format. This requires that the receiving site must either use the same format or convert it to their own. Difficulties can usually be avoided by reformatting the data to raster-scan blocking with pixel and block sizes that eliminate pad characters. Data accessing subroutines could also be transmitted with the picture.

- Add an additional header or directory file. The first header would contain fixed format transmission parameters and an index to the transmission contents. The index would enable programs to skip directly to the raw image data without parsing other header records. The headers and the image data may require additional padding to make each data section begin on a physical record boundary.

- Reformat the file. Both headers and image data could be reformatted to some agreed-upon transmission standard (such as the NATO RSG-4 9-track tape format). This may require significant conversion effort at both sites for block-scanned images, packed data, incompatible character sets, or other unsupported data structures.

Reformatting the file may be the only practical method if transmission via 7-track tape is required. A NATO RSG-4 7-track standard is available for this purpose. Machine-dependent integer or floating-point formats may also force reformatting. (The NATO formats support only ones's complement integers. Floating-point numbers are transmitted as signed integer exponent and mantissa fields each occupying an arbitrary number of bytes.)

Picture files written to magnetic tape for intersite transmission should have physical blocks no larger than 4096 characters. Header and trailer records should be padded to occupy an integral number of physical records. Note that this padding may require the header or trailer length fields to be changed.

Nine-track tapes should be either 800 BPI NRZI (which is the NATO standard) or 1600 BPI phase-encoded with odd parity and ASCII text. Seven-track tapes should be 800 BPI NRZI with even parity, and binary- coded decimal (BCD) characters. (Difficulties may arise in converting ASCII IMGDOC information to BCD. It is suggested that the IMGDOC records remain in ASCII during transmission, and that the receiving site then reformat the information as desired.)

Picture files for intersite transfer should not span multiple magnetic tapes. A picture too large to fit onto one tape should be broken into smaller pictures. More than one data file may be placed on a tape (separated by file marks), although some sites may require single-file tapes. The last data file must be followed by two consecutive tape marks.

It is recommended that this documentation file also be recorded on the tape if practical.

Potential Modifications

It is suggested that a "virtual picture" data type could be developed that would permit use of a window within a large picture file as if it were a separate picture file. The accessing information would usually be stored as a separate file or database entry. Every picture file would be described by at least one such descriptor, but multiple descriptors could refer to windows or bands within a single picture file. A descriptor could also define a composite picture made of several independent images (e.g., color bands stored in separate files).

The current picture format does not fix the lengths of header and trailer records. Either the size of all records could be fixed, or a fixed-length FILEHDR could be required in order to specify the lengths of following records. These restrictions will only be made if they prove necessary.

SEE ALSO

picnames(5), docformat(5), hdrformat(5)

BUGS

The FILEHDR has never been implemented. Header parsing code would be simplified if it were implemented as a separate file rather than a header record.

HISTORY

07-Feb-83 Laws at SRI-IU
Created.

NAME

picnames – standard picture file feature and property names

DESCRIPTION

Standard picture file feature names are given below; these filename extensions follow the CMU image naming convention. The conventions may or may not be enforced by any particular software system.

Names of properties (and their values when appropriate) for the CMU putprop() and getprop() routines are also given. These properties are intended to be stored in the picture data file as part of a header or trailer record.

FEATURE NAMES

red	Red band of a (usually) three-color image.
green	Green band of a three-color image.
blue	Blue band of a three-color image.
bw	Black and white image.
int	Intensity (density) computed from a multiple-band image.
hue	Hue (usually computed from a three-color (rgb) image).
sat	Saturation (usually computed from a three-color (rgb) image).
y	Color television y parameter.
i	Color television i parameter.
q	Color television q parameter.
cir	Color infrared band.
ohta2	Ohta's second-favorite feature (his first was int): red - blue.
ohta3	Ohta's third-favorite feature: (2 x red) - green - blue.
gradp	Gradient space p parameter.
gradq	Gradient space q parameter. Gradp and gradq have to do with hypothesized surface orientation.
mss4	Landsat MultiSpectral Scanner band 4.
mss5	Landsat MultiSpectral Scanner band 5.
mss6	Landsat MultiSpectral Scanner band 6.
mss7	Landsat MultiSpectral Scanner band 7.
sobel	Result of Sobel operator.
duda1	Result of Duda operator in the horizontal (x) direction.
duda2	Result of Duda operator at 45 degrees.
duda3	Result of Duda operator in the vertical (y) direction.
duda4	Result of Duda operator at -45 degrees.
gradx	Gradient of the intensity, x component.
grady	Gradient of the intensity, y component.

PROPERTY NAMES

pixel type

This property may have the values *unsigned*, *signed*, or *float*. If the value is *unsigned* (or if the property is not present), then the regular pixel-accessing routines should be used with the image. If the value is *signed*, then the sign-extending versions of the pixel-accessing routines should be used with it. If the value is *float*, then it should have 32 bits per pixel and should be accessed with a type cast like this:

foo = (float) pixel (image, row, column);

Note that there is a space in the property name "pixel type".

description
The value of this property is an informal description of the image. Programs should expect the value to be up to 150 characters in length.

BUGS

Filename extensions of more than three characters can be supported by UNIX and EUNICE systems, but are not currently supported by VMS. EUNICE (a UNIX emulation that runs under VMS) uses a special name translation system that generates two files with "hashed" names to represent the single data file and its name. This is transparent to the user unless VMS system utilities are also used.

SEE ALSO

cmuimglib(3), getprop(3), IUS Whitepapers CMU003, "CMU Image Format and Paging System," and CMU004, "Image File Naming Conventions."

HISTORY

28-Apr-81 Steve Clark (sjc) at Carnegie-Mellon University
Changed "signed" property to "pixel type" with various values.

27-Apr-81 Steve Clark (sjc) at Carnegie-Mellon University
Created.

PART 7 — Miscellaneous Information

NAME

tbhier – IU Testbed file system hierarchy

DESCRIPTION

The following gives a quick tour through the IU Testbed directory hierarchy.

/iu/tb/

This is the root of the current Testbed file system at SRI. The following are the major directories of programs and data which support the Testbed system:

bin bugs doc docsrc graphics include lib lisp man ms pic src

bin/ Testbed-specific executable programs, utilities, and shell scripts. The source code for all main programs in *bin/* is found in the *src/* directory. The documentation for each program or package is found in the *man/man1/* directory or in the *doc/* directory.

Major Testbed application programs in **bin/** are:

correlate camdist ghough line phoenix stereo showdtm

RELAX relaxation package programs include:

relax defcom defnbr imgprg prbimg relaxpar

Image processing utility programs are:

clip convert describe dpy erase gmrsys graphics imgsys invert normalize optronics overlay reduce shapeup show

The Testbed customized FRANZ LISP environment is **tblisp**.

General system utility programs and shell files are:

apropos catman ccr cpdir doc indent man mkwhatis.sh sc view whatis whereis whist

bugs/ text files describing known bugs and suggested system improvements.

doc/ major Testbed documentation text files.

docsrc/

photo-typesetter source files for Testbed documentation.

graphics/

source files for device-independent graphics test programs

include/

declarations and macro definitions used in Testbed C-language programs.

lib/ Testbed subroutine libraries and object module archives. These systems of subroutines are documented in the *man/man3/* directory.

- cmuimglib/ CMU-style image access utilities for reading Testbed image formats.
  - cmupiclib/ CMU routines to open and close picture files.
  - cmunmelib/ CMU image name parsing functions.
  - demo/ programs to exercise the cmuimglib routines.
  - piciolib/ CMU picture access routines.
  - tbpiclib/ Testbed picture access routines.
- cmuvsnlib/ CMU-style utilities needed to run CMU code "native."
  - convlib/ CMU image smoothing and convolution operators.
- devlib/ UNIX and VMS device driver sources.
- emacslib/ EMACS editor subroutine libraries.
- imagelib/ Testbed image and display software.
  - demo/ programs to exercise the imagelib routines.
  - dsplib/ Testbed display allocation.
  - frmlib/ Testbed picture frame system.
  - gmrlib/ Grinnell display device access code.
    - crslib/ cursor control functions
    - ctllib/ display control functions.
    - frmlib/ CMU frame system.
    - gmrlib/ miscellaneous Grinnell functions.
    - include/ declarations and definitions.
  - hdrlib/ picture header manipulation.
  - imgfrmlib/ utilities to display image files.
  - imglib/ IMG image structure manipulation.
  - imgnmelib/ image name parsing functions.
  - piciolib/ Testbed picture access functions.
  - piclib/ Testbed picture file manipulation.
- sublib/ local system utility subroutines.

arglib/	interactive Testbed argument parsing routines.
asklib/	CMU-style query routines used by CI driver.
blklib/	low-level blklib structure manipulation.
cilib/	CI command interpreter utilities.
doclib/	documentation record manipulation routines.
errlib/	Testbed error-handling code.
filelib/	file manipulation functions.
icplib/	utilities supporting the ICP command driver.
listlib/	list searching and manipulation functions.
lstlib/	LST list datatype routines.
mathlib/	mathematical functions.
matrixlib/	dynamic matrix manipulation routines
parselib/	argument parsing routines.
pntlib/	PNT point datatype routines.
seglib/	SEG line segment routines.
stringlib/	string manipulation functions.
syslib/	local operating system extensions.
timelib/	time and date utilities.
vectorlib/	vector manipulation functions.
wdwlib/	WDW window structure utilities.

visionlib/ high level machine vision subroutines.

blklib/	high-level blklib structure manipulation.
convlib/	PIC-based convolution routines.
histlib/	histogram manipulation routines.
intervlib/	histogram interval utilities (PHOENIX).
matchlib/	correlation and image matching routines.
patchlib/	patch and region utilities (PHOENIX).
polygnlib/	polygon manipulation utilities (PHOENIX).
relaxlib/	relaxation package support routines.
windowlib/	display window (DPY) manipulation subroutines
xformlib/	color coordinate conversion routines.

whatis database for the **whatis** command

lisp/ root directory for the Testbed **tblisp** FRANZ LISP system.

src/ sources for the Testbed FRANZ LISP routines in **tblisp**.

csrc/ sources for C-language programs loaded into **tblisp**.

lib/ standard LISP utility sources; this directory is aliased to */lisplib/*.

graphics/grinnell/
Grinnell graphics access from Lisp.

help/ help files needed by the **tblisp** help utilities.

man/ UNIX Testbed manual page files.

man1/ main program documentation. Program sources are generally in **src/** and executable images in **bin/**.

man3/ subroutine documentation. Subroutine sources are in **lib/**

man5/ data structure documentation.

man7/ miscellaneous documentation.

ms/ MAINSAIL program root directory. The root also contains the major executable image *mainsa.exe*, the system library *system.lib*, and the miscellaneous files needed to reconstruct them.

csrc/ C-language sources needed supporting the MAINSAIL system.

lib/ VMS-style Grinnell and picture libraries to link into MAINSAIL systems.

src/ MAINSAIL sources for the system.

pic/ the Testbed demonstration picture library. Each picture is assigned a unique subdirectory that contains image data and descriptive files. (e.g., *red.img*, *blue.img*, *green.img*, *pic.dat*). See also */aux/tbpic/*.

src/ root directory for the sources of the Testbed main programs and utilities. Executable images for these programs are in the *bin/* directory, while documentation is in *man/man1/*.

Major program directories include:
camdist correlate ghough line phoenix relax showdtm stereo

Image processing utility program directories are:
clip convert describe dpy erase gmrsys graphics imgsys invert normalize optronics overlay reduce shapeup show

General system utility program directories are:
catman ccr doc indent man sc whist view

Development in progress:
draw vshow

/iu/acronym/
root directory of the ACRONYM 3D object identification system.

compat/ compatibility packages for converting ACRONYM MACLISP source code to run under FRANZ LISP.

doc/ miscellaneous documentation for installers and implementers.

graphics/	graphics packages for ACRONYM displays.
info/	EMACS INFO nodes for interactive documentation.
models/	geometric model files.
sys/	executable image of acronym and all the source files required to build it.

/iu/usc/

preliminary version of Nevatia-Babu line finder, containing the following source and executable directories:

apar bin convolve distribution doc linkseg psmaker showdat thrin tst

/iu/info

Source of text information for the EMACS online information system.

/iu/testbed

Standard demonstration directory, also a generic user whose files can be customized for each new user.

demo/

standard demonstration directory. Names of directories often correspond to the standard demo image data. The file *demo* in each subdirectory is a shell file that runs the standard demonstration.

/aux/tbpic

auxiliary directory of image data. See the */iu/tb/pic* directory for frequently used demonstration images.

/aux/backup

this directory is used to back up programs in certain makefiles. If something goes wrong in the **make** procedure, the files can be recovered from this directory.

SEE ALSO

apropos(1), whatis(1), whereis(1), which (1), ncheck(8), find(1).

BUGS

The location of directories and files is subject to change without notice.