Image and Graphic Tools For Engineering Applications

Carl R. Crawford

Csuptwo, LLC

8900 N. Bayside Drive
Bayside, WI 53217-1911
Office/Fax: 414-446-4566
crawford.carl@csuptwo.com
www.csuptwo.com

Printed, September 14, 2011

Contributors

Mani Azimi
Ibrahim Bechwati
Owen Dake
Paul Granfors
Muzaffer Hiraoglu
Matt Hirsch
Jim Kohli
Greg Larson
Hara Levy
David Rozas
Chris Ruth
Malcolm Slaney

Table of Contents

NAME

crc - package of image, plotting and parsing tools

DESCRIPTION

This package of software includes tools for displaying images, plotting scientific data, parsing the command line passed to C programs,. The name of the package, **crc,** corresponds to the initials of the author of the software. The following programs and libraries are available:

anmi Annotates *mi*-files.

armi Archiver of *mi*-files.

asda Converts ASCII files to *da*-files.

cda Combines *da*-files.

cmi Combines *mi*-files.

daas Converts *da*-files to ASCII files.

dami Converts *da*-files to *mi*-files.

dft Takes the DFT (discrete Fourier transform) of *da*-files.

exda Extracts records from *da*-files.

fmm Finds maximum and minimum in *da*-files.

lmmi Converts *lm*-files to *mi*-files.

mag Magnifies (2X) *mi*-files.

minda Minifies *da*-files.

sagcor Generates sagittal, coronal, and projection images from *mi*-files.

xpic Displays images in X11 window.

lc Extracts lines/columns from **xpic**(1).

cplot Contour plotter.

plot3d Three-dimensional plotter of elevation data.

plotps Converts **plot**(5) format to Postscript.

qplot Quickly plots vectors.

xpc Controls **xplot**(1) windows.

xplot Displays **plot**(5) commands in X11 window.

[lib]crcplot.a
Graphics library.

[lib]parse.a
C support library. Reads and writes *da-files* and *mi-files.* Parses command line.

PHILOSOPHY

This package of tools is optimized for scientific and engineering applications. Principle tools are used for image display and for graphics. Other tools are available for developing C programs, generating common image and data formats, and converting to and from the common formats. Note, all image and graphics displays are separated from the scientific or engineering application. Application programs communicate to the displays using standardized image and vector formats. Additionally, the operating system is hidden from user via calls to a support library. Using these tools, the scientist or engineer can concentrate on development instead of on graphics, image display, and the operating system. Following are descriptions of the key tools.

xpic displays images in the X11 windowing environment. Sixteen bit images are displayed in a user-specified region of a canvas. The program is optimized for an format called *mi- files.* Support for this format is provided in a support library. Window/level functionality, from 16 to 8 bits, is provided via sliders.

Commands are entered in a *TTY* window. Images can be annotated with text or cursors. The canvas can be converted to PostScript or sent to a laser camera. Some simple image processing - offset, scale, and region-of-interest analysis - is available. Complex image processing is not available with the program because it would violate the philosophy used for the package. Applications that require interactive use of an image display can use a library of functions that communicate with the program using inter-process communication. In particular, shared-memory access is available to the 16 bit image memory.

xplot displays Unix *plot* commands in the X11 windowing environment. It listens for these commands through inter-process communication. A library of functions is provided to draw complex graphical objects like axes, grids, and text using the *plot* format. The library also has functionality to connect the application program to *xplot.* The programs *qplot, plot3d,* and *cplot* use the library to plot vectors (x versus y), elevation maps, and contour plots, respectively, from data contained in common formats. The preferred format is another format called *da- files.* Support for this format is also provided in the support library. Therefore, the user can either embed the graphics in an application using library calls or write data to external files and use the three plotters for subsequent visualization.

parse.a is the support library. In addition to handling *da-* and *mi- files,* functions are provided for passing parameters on the command line and for hiding specifics of Unix. The latter feature is useful for migrating applications to different platforms. Error detection and reporting is built into all the functions.

INSTALLATION

The code is distributed in either **crc.tar** or **crc.tar.Z.** If necessary, uncompress the latter file using **uncompress**(1). Unwrap the code using **tar**(1) and make the code by running **make**(1) in the directory **./crc.** Some editing of the *Makefile* might be necessary to reflect your environment. Read the file **crc/README** for additional installation details. By default, the *Makefile* is setup for Linux. Compilations for Solaris are also supported by uncommenting and commenting a few lines of the *Makefile.*

SUPPORT

Please report all bugs and comments to the author listed below. If you would like to receive updates and status reports on the code please let the author know through electronic mail and you will be added to a distribution list.

COMMAND LINE PARSING

The programs in this package share a common parser to extract options from the command line. The purpose of this section is to provide a detailed description of the operation of the parser. This section is organized as follows. First, an overview of the parser is presented. Then, each of the basic parts of the parser are described in detail. For additional information on the incorporation of the parser in a program, the reader is referred to **parse_parse**(3).

Programs using the parser adhere to the Unix philosophy that the program will do something useful when invoked without specifying any options on the command line. For the sake of discussion, consider a program named **prog.** The following line invokes the program without any specified options

% prog

where percent (%) is the default prompt from the Unix shell. In response to this command, the program might read information from a file or the terminal and probably output information to a file or on the terminal.

The parser processes two types of objects: flags and options. Flags are the individual characters following a minus (-) sign. Examples of flags are **-a** and **-abc,** where each of the letters are individual flags. In this case, **-a, -b,** and **-c,** are the flags. Options are words, followed by an equals sign, and then followed by a value. Examples of options include **infile=filename** and **weight=10,** where **infile** and **weight** are the option names, and *filename* and *10* are the values of the options. For example, the program **prog** could be invoked with options and flags as follows

% prog infile=input_file -a -bc weight=10

where the input would be taken from the file *input_file,* a weight of *10* would be applied instead of a default weight, and program modes associated with **-a, -b** and **-c** would be used.

The parser also allows the specification of flags and options in initialization files and in environment

variables. The names of the files and the variables are program dependent. For the sake of discussion, assume that the sample program **prog** uses the file **.prog** and the environment variable **PROG.** Please refer to the documentation for your specific shell to determine how to set environment variables. The program **prog** examines **.prog** in the user's HOME directory, and then the program examines **.prog** in the current working directory, if it is not the HOME directory. Next, the program examines the variable **PROG.** Finally, the command line is examined. The contents of **.prog** might be

-a weight=10
-b

and **PROG** might be

-c infile=results

Now consider the execution of the program without any options or flags as in

% prog

Because of the existence of **.prog** and **PROG,** the command line execution of the program is equivalent to

% prog -a weight=10 -b -c infile=results

Specification of the flag **-@** disables the use of initialization files and environment variables.

The formal definition of a flag is a string beginning with a minus sign (-) and followed by a set of characters. Each of the characters following the minus sign are known as flags. At least one flag must be present after the minus sign. The parser maintains a list of valid flags and actions to take when a flag is encountered. In all cases, flags consist of only one character. Typically, a variable is set to one when the flag is present on the command line. The string of flags usually consists of upper- and lower-case letters. However, other characters or numbers can be used. Be aware that some characters (for example, the question mark) might have to be escaped to prevent interpretation by a shell using backslashes or quotation marks. The question mark, when it is part of a flag, signifies that help is requested (more information is provided below on its use). Examples of valid flags are

-a
-Bcd
-?

The grammar is set up so that flags are processed from left to right on the command line. Therefore,

% prog -ab

is equivalent to

% prog -a -b

The effect of some flags can be negated, where negation means that flag will be effectively unset. When a minus sign (-) is encountered in a list of flags, all flags afterwards will be negated. When a plus sign (+) is encountered, negation is turned off. In the example

% prog --bc

the flags **-b** and **-c** are unset (i.e., negated). In the following example

% prog -a-bc+de

the flags **-a, -d** and **-e** are set and flags **-b** and **-c** are unset.

If an illegal flag is specified, then the following error message is displayed and the program halts execution

bad flag: -k

where **k** is the illegal flag.

As mentioned above, the flag **-@** disables the use of initialization files and environment variables.

An option has the form **name=value,** where **name** is the name the of the option and *value* is a number or string to be assigned to **name.** The parser maintains a list of options and actions to take if the option is present on the command line. Typically, a value is assigned to a variable, or a string is copied to a buffer.

A common use of options is to specify the names of input and output files. The parser allows the **name** and *value* fields to be two separate words with the equals sign (=) still attached to the *name* as in the following example

% prog infile= ../tmp/junk.txt

This functionality is provided so that filename completion can be used with the shell. A single question mark, **?,** or a question mark with an equals sign, **?=,** indicates that help is requested (more details below).

Options can be abbreviated to the fewest number of characters that will not cause ambiguity in the parser. Consider a program that has options **xoffset, yoffset, of,** and **offset.** Then, the options **x** and **xo** are equivalent to using **xoffset.** If **o** was used on the command line, then the parser would report the following error:

Abbreviation, o, matches: of offset

The parser does not report an error in the case if **of** because **of** matches the complete spelling of an option.

If an illegal option is specified, then the following error message is displayed and the program halts execution

Illegal option: name

where **name** is the illegal option. If the value for an option is missing, then the following error message is printed

empty string or no next arg: option

Many of the programs allow one option to be specified without using its name or an equals sign. This mode is useful when one value is almost always specified on the command line. An example is the plotting program **qplot**(1) which has the option **y** to specify a vector to plot. To plot the vector **data.da,** either of the following can be used

% qplot y=data.da
% qplot data.da

Some options, specifically text fields, can contain spaces and other characters that are trapped by the shell. In these cases, the value given to the option should be protected using quotation marks. For example, **qplot** uses the option **yl** to specify the label below the y-axis. A label with spaces and asterisks would be specified as follows

% qplot yl="*label with spaces and asterisks***"**

Options that receive numeric arguments can be passed an arithmetic expression. The expression can contain numbers, including those in exponential format (i.e., those with 'e' or 'E'), '+' for addition, '-' for subtraction or unary minus, '*' for multiplication, and '/' for division. The priority convention for the operations is the same as C or Fortran, and can be changed with the use of parentheses. Note that no spaces can be in the expression and the use of quotes might be required to protect the expression from the shell. Complete details on expressions can be found in the documentation for **parse_atof**(3). Examples of the use expressions include the following

% prog weight=10+20
% prog weight="1e2+20"
% prog weight="1e2+20"
% prog weight="(8+2)*(30+10)/2"

where **weight** is assigned a value of 30 in all four cases.

Command line arguments can be placed in an initialization file. The name of the file is unique to each program. The file in the user's HOME directory is first examined, if it exists. If the current working directory is not the HOME directory and the file exists in the current working directory, then the file is examined in the current working directory. Command line arguments are read from the files as if they were present on the command line. Arguments are read until the end-of-file or until a line beginning with // is encountered. Lines beginning with a sharp (#) are comments. Any text after a sharp, including the sharp itself, is also a comment and is stripped off. An arbitrary number of arguments can appear on the same line. By convention, the name of the initialization file is the name of the program with a period (.) prefixed and perhaps

with the suffix **rc** appended. A sample initialization file for the program **xpic**(1) is as follows

```
#       .xpic - startup file for xpic
-a              # don't allow xpic_attach
-p              # enable port access
#-o             # disable options panel
window=300      # default window
level=0         # default level
#       commands after // are "entered" at xpic prompt
//
file pic.mi,slick.mi # load pic and slick side-by-side
```

Options and flags can also be specified in environment variables. The mechanism to set these variables is specific to each shell. For example, the variable **PROG** for the sample program **prog** can be set for the Bash shell using

% export PROG="weight=10 -a"

Initialization files are examined first. Then the environment variable is parsed. Finally, the command line is processed. The flag **-@** disables the use of initialization files and environment variables.

The command line is parsed from left to right. For the sake of this discussion, it is assumed that parameters in files and in the environment are part of the command line. In general, the parser uses the last value of an option or the mode specified by a set of flags. Consider the following case with options

% prog weight=10 weight=20

Then the value *20* is used in the program. Now assume that flags **-a** and **-b** enables modes "a" and "b", respectively, where the modes are opposites of each other (i.e., only one of the two modes can be active in any given invocation of the program). Then

% prog -a -b -a -a -b

invokes mode "b". The use of multiple specifications is useful to override parameters in files or the environment, or in conjunction with the history mechanism of most Unix shells. Note that flags and options can be mixed on the command line as in the following example

% prog -a weight=3.14159 -c

The parser allows the automatic setting, which is called side effects, of an option when a flag is set or a flag when an option is set. The biggest use of side effects is to automatically set program modes associated with certain options. For example, assume that the program has the two modes of operation "a" and "b," which are associated with the flags **-a** and **-b.** Also assume that mode "b" uses an option **bfac=** The brute force way to invoke the program in the "b" mode with a specified value of *bfac* is

% prog -b bfac=10

Since **bfac** is used only with the "b" mode, the program automatically invokes the **-b** flag. Therefore, the following command line is equivalent to the previous line

% prog bfac=10

When command lines are parsed, the user has to be aware of side effects that might have to be overridden.

A program's command line syntax is written to the user's terminal if one of the following flags or options is present on the command line: **-?, ?=** or **?.** Be aware that the question mark might have to be escaped to prevent interpretation by a shell using backslashes or quotation marks. The parser lists valid flags and options. In the case of options, the default value of the option is presented. The option **(empty)** in the help screen is the option that can be invoked without an equals sign. Here is the help screen for **cda**(1)

```
-p: don't print program information
-P: prompt for parameters
-a: add vectors
-s: subtract vectors (default)
-m: multiply vectors
```

-d: divide vectors
i1=y1.da: first input file
i2=y2.da: second input file
of=y.da: output file
w1=1.000000: weight for first vector
w2=1.000000: weight for second vector
(empty) equivalent to: set files

FILES

crc.tar	Uncompressed **tar**(1) file in which the package is distributed.
crc.tar.Z	Compressed **tar**(1) file in which the package is distributed.
crc/Makefile	Master *makefile* for the package.
crc/README	Contains detailed notes on the installation of the package.

SEE ALSO

anmi(1), **armi**(1), **asda**(1), **cda**(1), **cplot**(1), **daas**(1), **dami**(1), **exda**(1), **fmm**(1), **lc**(1), **lmmi**(1), **mag**(1), **minda**(1), **sagcor**(1), **plot3d**(1), **plotps**(1), **qplot**(1), **xpc**(1), **xpic**(1), **xplot**(1), **axis**(3), **clear**(3), **crcplot**(3), **curve**(3), **dline**(3), **factor**(3), **grid**(3), **hardcopy**(3), **hatch**(3), **line**(3), **number**(3), **plot**(3), **plotcrc**(3), **plots**(3), **scale**(3), **symbol**(3), **where**(3), **example**(8), **parse**(3), **parse_accept**(3), **parse_atof**(3), **parse_buffer**(3), **parse_canonical**(3), **parse_check**(3), **parse_da**(3), **parse_disk**(3), **parse_fft**(3), **parse_malloc**(3), **parse_mi**(3), **parse_parse**(3), **parse_print**(3), **parse_string**(3)

AUTHOR

Carl R. Crawford

NAME

anmi - annotate mi-files

SYNOPSIS

anmi *[options]*

DESCRIPTION

By default, **anmi** adds the label "anmi" to the header of the last image contained in the mi-file named *pic.mi.*

The following options are available to change the default performance of the program:

if=*file[.mi]* The mi-file named *file[.mi]* is used instead of the default file *pic.mi.* The suffix *mi* is added if necessary.

label=*text*

text The specified text is used as a label. If an equals sign is present in the label, then the **label=** prefix must be used. The default text is "anmi".

-l The last image is annotated. This is the default.

-f The first image is annotated.

n=*n* Image number *n* is annotated, where n=1 is the first image.

-p The program will not print out information and therefore runs silently.

SEE ALSO

parse(3), **parse_mi**(3), **xpic**(1), **armi**(1), **asda**(1), **cda**(1), **daas**(1), **dami**(1), **exda**(1), **fmm**(1), **minda**(1), **mag**(1)

AUTHOR

Carl R. Crawford

NAME

armi - archive mi-files

SYNOPSIS

armi c|d|r|t|x*[n][v]* archive*[.mi] [num] file[.mi]...*

DESCRIPTION

armi maintains "archives" of *mi*-files. This program allows the user to add, extract or delete images in an existing *mi*-file. A new *mi*-file can be created from a set of existing *mi*-files. Finally, the contents of an existing archive can be displayed. The operation of the program closely follows the program **ar**(1). In all cases, the name of the resulting archive is given by *archive[.mi].* The suffix *.mi* will be added if necessary. One of the following operation modes must must specified:

c The images contained in the files *file[.mi]...* are combined to form a new (or created) archive. The archive will be deleted if it already exists.

r The images contained in the files *file[.mi]...* are appended to an existing archive. The archive will be created if necessary. Old style *mi*-files cannot be used for this operation; see **parse_mi(3)** for additional details.

d The images specified in slots given by *files...* are deleted from the archive. In this mode the files are actually numbers. The first image in an archive is number one. The images left in the archive are renumbered. The archive is deleted if all the images are deleted.

x By default, the first image in the archive is written to the first file specified on the command line. The second image is written to the second specified file and so forth. If the **n** option is present, then the first image to be extracted is given by *num* instead of image one. Even though the image is extracted from the archive and placed in a new file, it remains in the archive. The function **parse_wmi**(3) is used to write the new images. Therefore, double underscores (__) can precede the new filenames to to append the extracted images to existing *mi*-files.

t The contents of the archive are listed. If the *v* flag is specified, then additional (verbose) information about the format of the *mi*-file will be supplied; see **parse_mi(3)** for additional details about the supplied information.

FILES

armi.0x3x5x.mi Temporary file used by the **d** option.

SEE ALSO

armi(1), **dami**(1), **mag**(1), **parse**(3), **parse_mi**(3), **xpic**(1)

AUTHOR

Carl R. Crawford (ccrawford@analogic.com)

BUGS

Labels formed from input filenames will be silently truncated if they are too long.

NAME

asda - convert ASCII files to da-files

SYNOPSIS

asda *[options] [file[.as]]*

DESCRIPTION

By default, **asda** converts the numbers in ASCII format contained in the file *y.as* and outputs a *da*-file named *y.da* containing the numbers. The input file consists of multiple lines. Each line can contain a maximum of two hundred values in "free format". In other words, spaces, tabs and newlines can all be used to separate the numbers. Lines are read until and end-of-file is reached. By default, the first number on each line is used.

The numbers read are called the y-vector. Options exist to convert a second vector, called the x-vector, from the input file.

The following options are available to change the default performance of the program:

if=*file[.as]*

file[.as] Data are read from *file[.as]* instead of the default file *y.as.* The suffix *as* is added if necessary.

of=*file[.da]* The image is written to *file[.da].* The suffix *da* is added if necessary. The default output file is the name of the input file with the suffix *da* added. If a suffix is already present in the input filename, it is first deleted.

ny=*col* The y-vector will be read from column *col* instead of column one. If the **nx** option is used (see below) then the column number is relative to the x-column number and *col* can be negative.

nx=*col* A x-vector is read from column *col* in addition to the y-vector. The suffix *_x.da* is added to the output filename to form the name of the file containing the x-vector.

count=*c* Read only *c* points from the input file. By default, all the points are read until an end-of-file is reached.

skip=*n* Read only every *n*'th plus 1 point from the input file. The default value is zero.

begin=*b* Begin reading the input file at line *b,* where the first line in the file is line zero.

-p The program will not print out information and therefore runs silently.

SEE ALSO

qplot(1), **parse**(3), **parse_da**(3), **xpic**(1), **armi**(1), **asda**(1), **cda**(1), **daas**(1), **dami**(1), **exda**(1), **fmm**(1), **minda**(1), **mag**(1)

AUTHOR

Carl R. Crawford

NAME

cda - combine da-files using mathematical operations

SYNOPSIS

cda *[options] [file1[.da] file2[.da]]*

DESCRIPTION

By default, **cda** subtracts the data contained in a second file *y2.da* from the data contained in a first file *y1.da* and writes the resulting data into *y.da.*

The following options are available to change the default performance of the program:

i1=*file[.da]* The first file is set to *file[.da]* instead of the default *y1.da.* The suffix *da* is added if necessary.

i2=*file[.da]* The second file is set to *file[.da]* instead of the default *y2.da.* The suffix *da* is added if necessary.

file1[.da] file2[.da]
The first and second files are set to *file1[.da]* and *file2[.da],* respectively, instead of *y1.da* and *y2.da.* The suffix *da* is added if necessary.

of=*file[.da]* The image is written to *file[.da].* The suffix *da* is added if necessary. The default output file is *y.da.*

-s The data are subtracted, which is the default mode of operation.

-a The data are added.

-m The data are multiplied.

-d The data are divided.

w1=*w* The data in file one are weighted by *w* before the specified operation is performed. The default value of the weight is one.

w2=*w* The data in file two are weighted by *w* before the specified operation is performed. The default value of the weight is one.

-p The program will not print out information and therefore runs silently.

SEE ALSO

qplot(1), **parse**(3), **parse_da**(3), **xpic**(1), **armi**(1), **asda**(1), **cda**(1), **daas**(1), **dami**(1), **exda**(1), **fmm**(1), **minda**(1), **mag**(1)

AUTHOR

Carl R. Crawford

NAME

cmi - combine mi-files using mathematical operations

SYNOPSIS

cmi *[options] [file1[.mi] file2[.mi]]*

DESCRIPTION

By default, **cmi** subtracts the data contained in a second file *in2.mi* from the data contained in a first file *in1.mi* and writes the resulting data into *out.mi.* The minimum and maximum values in the two input files and the output file are also reported.

The following options are available to change the default performance of the program:

i1=*file[.mi]* The first file is set to *file[.mi]* instead of the default *in1.mi.* The suffix *mi* is added if necessary.

i2=*file[.mi]* The second file is set to *file[.mi]* instead of the default *in2.mi.* The suffix *mi* is added if necessary.

file1[.mi] file2[.mi]
The first and second files are set to *file1[.mi]* and *file2[.mi],* respectively, instead of *in1.mi* and *in2.mi.* The suffix *mi* is added if necessary.

of=*file[.mi]* The image is written to *file[.mi].* The suffix *mi* is added if necessary. The default output file is *out.mi.*

-s The images are subtracted, which is the default mode of operation.

-a The images are added.

w1=*w* The data in file one are weighted by *w* before the specified operation is performed. The default value of the weight is one.

w2=*w* The data in file two are weighted by *w* before the specified operation is performed. The default value of the weight is one.

offset=*o* The offset *o* is added to the result. The default value of the offset is zero.

-o Only one file is processed. This means that the pixels are multiplied by *w1* and the offset *offset* is added.

-p The program will not print out information and therefore runs silently.

SEE ALSO

qplot(1), **parse**(3), **parse_da**(3), **xpic**(1), **armi**(1), **asda**(1), **cda**(1), **cmi**(1), **daas**(1), **dami**(1), **exda**(1), **fmm**(1), **minda**(1), **mag**(1)

AUTHOR

Carl R. Crawford

NAME

cplot – generate contour plots

SYNOPSIS

cplot [*options*] [*file[.da|.mi|.as]*]

DESCRIPTION

cplot generates contours of a three dimensional surface. The resulting graphics is sent directly to the X-window, called a plotting window, maintained by **xplot**(1). **xplot**(1) has to be running before executing **cplot**. **xplot**(1) and **cplot** can be run on different hosts if the environment variable **XPLOTHOST** is used.

A matrix of binary floating point numbers will be read from a da-file named *z.da* and used as the z value at each point. This vector will be displayed against a program generated xy-matrix containing integers from zero to the number of points in each direction minus one. The points in the file are assumed to be rows of values along lines of constant y.

The input file can be either a binary or an ASCII file. In a binary file, the data is coded in big-endian format. The file can be generated using the **fwrite**(3s) statements in C or an unformatted write in Fortran or Pascal. This is the most efficient form since it saves on both file space and machine time. *mi*-files, containing images, and *da*-files, containing data with a header, can also be used as input. See **parse_da**(3) and **parse_mi**(3) for information on these file formats.

For simple applications it is also possible to use an ASCII file with the numbers in a readable format. The user must be careful to edit out any titles or other non-numeric information. The numbers are read from the file in "free format". In other words, spaces, tabs and newlines can all be used to separate the numbers. The program will read as many lines as necessary to get enough data for the plot as specified by the **count**, **skip** and **begin** options (see below for additional information). The numbers that are read from the file must not contain any spaces.

The program supports as many of the **qplot**(1) options as possible and in most cases the options/flags are known by the same name. The most significant difference is that the data file is indicated by the **z** option instead of the **y** option for obvious reasons.

The input data are a matrix of binary numbers. The type and location of the data is specified with the **z** option. The matrix is considered to be size 64 x 64 unless one of the size options (**size**, **xsize**, or **ysize**) is used. The use of *mi*- or *da*-files overrides this assumed size.

Data are read from the file along lines of constant y (x variable varies fastest). The first value in the file, file(0), is z(x=0, y=0), the second is z(x=1, y=0) and so on. The second line (y=1) starts at number **xsize** in the file (file(**xsize**) --> z(x=0, y=1)). It is very important that the variable **xsize** be set properly so that the program can index into the matrix properly.

With the **x** and **y** options an arbitrary mapping can be made between points in the matrix (z) and the x and y axis. Normally each line of the data is assumed to be equally spaced in x and y. With the **x** and **y** options any arbitrary mapping is possible. The results are guaranteed to be meaningful only for monotonically increasing mappings.

With the **begin**, **skip** and **count** parameters it is possible to specify that part of the input data be ignored. If the **begin** options are used, the first **xbegin** data points of the z (per row) and x files are ignored. A similar effect is seen if the **ybegin** parameter is specified. The default values are 0. If the **skip** options (**skip**, **xskip**, and **yskip**) are set then lines in the input matrix are skipped between each point that is plotted. The **count** option is used to set the number of data lines to plot. Normally the **count** parameters default to the largest number possible, given the **size**, **begin** and **skip** variables.

OPTIONS

The following options and flags are available on the command line to modify the performance of the program. Options can be minimally abbreviated.

****** FILE SPECIFICATION OPTIONS ******

z=*file[,n]* The z matrix will be read from *file* instead of *z*. The letter *n* is the data type declaration field and can be one of the following:

- **c** Single bytes, unsigned, fixed precision **char** data.
- **cs** Single bytes, signed, fixed precision **char** data.
- **s** Two bytes, signed, fixed precision short **int** data.
- **i** Four bytes, signed, fixed precision **int** data.
- **l** Four bytes, signed, fixed precision **long int** data.
- **f** Four bytes, **float** data.
- **p** Eight bytes, **double** floating-point data.
- **a** ASCII data, free format numbers, data is readable, spaces, tabs and newlines are used to separate input points.
- **d** *da*-file, see the appendix of **parse_da**(3) for file format.
- **m** *mi*-file, see the appendix of **parse_mi**(3) for file format.

If *n* isn't specified, a comma is not needed after the file name. The default for *n* is **d**.

z=,*n* The z vector is read from the default file *z* but the byte declaration field is set to *n*.

-*x* Read the x vector from the file *x*. The data in the file is assumed to be a da-file.

x=*file[,n]* The x vector will be read from the file *file* instead of *x*. The data in the file is assumed to be a da-file, unless a *,n* is added. The variable *n* can be any of the suffixes shown above with the **z** option.

-**y** Read the y vector from the file *y*. The data in the file is assumed to be a da-file.

y=*file[,n]* The y vector will be read from the file *file* instead of *y*. The data in the file is assumed to be a da-file, unless a *,n* is added. The value of *n* can be any of the suffixes shown above with the **z** option.

xsize=*n* The matrix has *n* elements in the x-direction. The default value is 64. It is necessary to specify this variable if the array is not 64 x 64. The program interprets the z-file as an **xsize** by **ysize** matrix of values. It is not necessary to specify **xsize** or **ysize** if the input format is *da*-files or *mi*-files.

ysize=*n* The matrix has *n* elements in the y-direction. The default value is 64. The program interprets the z-file as an **xsize** by **ysize** matrix of values. It is not necessary to specify **xsize** or **ysize** if the input format is *da*-files or *mi*-files.

size=*n* Set both **xsize** and **ysize** equal to *n*.

n=*n* This sets the **xsize** and **ysize** variables equal to *n*. This is equivalent to using the **size** option.

****** DATA SELECTION OPTIONS ******

xbegin=*n* The first *n* columns of the input in the z-file (per row) and *n* values in the x-file are skipped. This is used to place the origin of the plot at an arbitrary position in the matrix. The default value is 0.

ybegin=*n* The first *n* rows of the input in the z-file and *n* values in the y=file are skipped. This is used to place the origin of the plot at an arbitrary position in the matrix. The default value is 0.

begin=*n* Set the parameters **xbegin** and **ybegin** equal to *n*.

xskip=*n* Skip *n* columns in the z-file (per row) and *n* values in the x-file between plotted data points. The default value is 0.

yskip=*n* Skip *n* rows in the z-file and *n* values in the y-file between plotted data points. The default value is 0.

skip=*n* Set the parameters **xskip** and **yskip** equal to *n*.

xcount=*n* Only plot *n* of the columns in the matrix. Show *n* values in the matrix for each line of constant y. The default value is the largest number of points that can be plotted for the given **xsize**, **xbegin** and **xskip**.

ycount=*n* Only plot *n* of the rows in the matrix. Show *n* values in the matrix for each line of constant x. The default value is the largest number of points that can be plotted for the given **ysize**, **ybegin** and **yskip.**

count=*n* Set the parameters **xcount** and **ycount** equal to *n*.

zmin=*r* All points in the matrix below the level *r* are set to *r*. This is useful for seeing details in the surface. See also the **zmax** option. The default value is obtained from the data itself.

zmax=*r* All points in the matrix above the level *r* are set to *r*. This is useful for seeing details in the surface. See also the **zmin** parameter. The default value is obtained from the data itself.

xmin=*min* Normally the x-axis is annotated with values from zero to the number of y lines minus one. With this option the minimum value on the x-axis is set to *min.*

xmax=*max* Only if used with the *xmin* option, floating-point annotations will be used between the value set with this option, *max*, and the value set with the **xmin** option.

ymin=*min* Normally the y-axis is annotated with values from zero to the number of x lines minus one. With this option the minimum value on the y-axis is set to *min.*

ymax=*max* Only if used with the **ymin** option, floating-point annotations will be used between the value set with this option, *max*, and the value set with the **ymin** option.

****** PLOT POSITIONING AND REDUCING OPTIONS ******

xp=*r* The x starting coordinate of the axes is moved to position *r* on the plot plane. The default value of *r* is 0. The plot is 10 units wide.

yp=*r* Same as the **xp** option but the y origin is moved to *r* on the plot plane. The plot is 10 units high.

scfac=*r* The graph is expanded or reduced in size by *r*. The default value is 1.0. The option **scale** is a synonym for this parameter. There is another scale factor of two built into the program.

-**e** Do not erase the **xplot**(1) window before plotting. This is useful for overlaying multiple vectors on the same set of axes.

-o Send the **plot**(5) format output to **stdout** instead of via a socket connection to **xplot**(1).

host=*[host][:socket]*
Connect to the plotting window on the machine named *host* using socket number *socket.* If the host is not specified, then the plotting window on the local host will be used. If the host is the string *stdout* the **plot**(5) commands will be sent to standard output. If the socket is not specified, then the socket listed in the section INTERPROCESS COMMUNICATION will be used.

****** AXIS OPTIONS ******

-**a** No axes will be plotted.

-**f** A border will be drawn around the plot.

len=*r* Set the length of the axes to *r*. The default value is 3.

xlen=*r* The length of the x axis is changed to *r* units.

ylen=*r* The length of the y axis is changed to *r* units.

-**g** Superimpose a grid over the plot. The grid connects the tic-marks on the axes to each other through a set of orthogonal lines.

****** LABEL OPTIONS ******

tl=*str* The string *str* will be used as a label at the top of the graph.

bl=*str* The string *str* will be used as a label at the bottom of the graph.

xl=*str* The string *str* is used as a label along the x axis.

yl=*str* The string *str* is used as a label along the y axis.

****** GRAPH OPTIONS ******

contours=*r* The number of contours is set to *r.* The default value is 10. The contours are set at equal increments between **zmin** and **zmax**. The option **c** is a synonym for this parameter.

t=*r* One contour will be drawn for the z value of *r* .

i=*r* Interpolate the data so that *r* extra points are inserted between each pair of data points in the *x* and *y* directions. This option can be used to smooth out the contours when the size of the surface is small. The default value is zero and the maximum value is ten.

ix=r Same as the **i** option but now only the interpolation factor for the x-axis is set.

iy=r Same as the **i** option but now only the interpolation factor for the y-axis is set.

DEFAULTS

The internal parse routines allow for two default mechanisms to specify options. The first method is to create files **.cplot** in your **HOME** directory and/or **.cplot** in the working directory. In them, one can place options and flags that will be used before parsing the command line. Lines beginning with a '#' are

considered to be comments. A '#' also marks the beginning of a comment anywhere else on an input line. Parsing of arguments ends when either an end-of-file is reached or when the line beginning with the string "//" is found. The second method to set default options is to use the environment variable **CPLOTARGS.** The format is the same as the input command line. The order or parsing is: **~/.cplot ./.cplot, CPLOTARGS,** and finally the command line. The flag **-@,** if present on the command line, will disable the use of the default mechanisms.

FILES

~/.cplot

./.cplot initialization (startup) files.

ENVIRONMENT

CPLOTARGS contains command line arguments.

HOME contains the shell's concept of your home directory.

XPLOTHOST Can be used to specify the host and the socket for the plotting window. They have the form *[host][:socket].* The first one is used if both are set.

INTERPROCESS COMMUNICATION

8124 Default socket used to talk to plotting window.

SEE ALSO

qplot(1), **crcplot**(3), **symbol**(3), **scale**(3), **parse_da**(3), **parse_mi**(3), **parse**(3), **parse_parse**(3), **parse_canonical**(3), **plot**(5), **plot3d**(1), **plotps**(1), **xpic**(1), **xplot**(1)

AUTHOR

Carl R. Crawford

cplot is based upon **plot3d,** which is based upon **qplot.** Malcolm Slaney developed **plot3d** based on **qplot.**

BUGS

No check is made to insure that the number of points to read from the input file for each line (**count***(**skip**+1)+**begin**) is consistent with the **size** options.

NAME

daas - convert da-files to ASCII

SYNOPSIS

daas *[options] [file[.da]]*

DESCRIPTION

By default, **daas** converts the *da*-file named *y.da* into ASCII format and writes the resulting text on **stdout.**

The following options are available to change the default performance of the program:

if=*file[.da]*

file[.da]	Data are converted from *file[.da]* instead of the default file *y.da.* The suffix *da* is added if necessary.
of=*file[.as]*	The image is written to *file[.as].* The suffix *as* is added if necessary. The default output file is **stdout.**
-n	The data are numbered.
-p	The program will not print out information and therefore runs silently.

SEE ALSO

qplot(1), **parse**(3), **parse_da**(3), **xpic**(1), **armi**(1), **asda**(1), **cda**(1), **daas**(1), **dami**(1), **exda**(1), **fmm**(1), **minda**(1), **mag**(1)

AUTHOR

Carl R. Crawford

NAME

dami - convert da-files to mi-files

SYNOPSIS

dami *[options] [file[.da]]*

DESCRIPTION

By default, **dami** converts the *da*-file named *y.da* into a *mi*-file and writes it to a file named *y.mi.* The resulting image is offset so that zero corresponds to 512. The data are scaled so that the values in the image lie between zero and 1023.

The following options are available to change the default performance of the program:

if=*file[.da]*

file[.da]	Data are converted from *file[.da]* instead of the default file *y.da.* The suffix *da* is added if necessary.
of=*file[.mi]*	The image is written to *file[.mi].* The suffix *mi* is added if necessary. The default output file is formed by replacing the *da* suffix in the input file with *mi.*
-p	The program will not print out information and therefore runs silently.
scale=*s*	The range of the output data is set to *s.* The default value, as noted above, is 1023.
zero=*z*	The zero (mid-point) of the output data is set to *z.* The default value, as noted above, is 512.
min=*x*	The value *x* is used instead of the minimum datum.
max=*x*	The value *x* is used instead of the maximum datum.

SEE ALSO

qplot(1), **parse**(3), **parse_da**(3), **xpic**(1), **armi**(1), **asda**(1), **cda**(1), **daas**(1), **dami**(1), **exda**(1), **fmm**(1), **minda**(1), **mag**(1)

AUTHOR

Carl R. Crawford

NAME

dft - takes the discrete Fourier transform (DFT) of da-files

SYNOPSIS

dft *[options]* [options/flags] [file[.da]]

DESCRIPTION

By default, **dft** takes the DFT of the each of the rows of data contained in the file *y.da.* The DFTs are written to the file *z.da.* The number of output points is the number of points per row divided by two.

The following options are available to change the default performance of the program:

if=*file[.da]* The input data is read from *file[.da]* instead of the default *y.da.* The suffix *da* is added if necessary.

of=*file[.da]* The DFT is written to *file[.da].* The suffix *da* is added if necessary. The default output file is *z.da.*

-l The logarithm of the DFT is written to file *zl.da.*

lf=*file[.da]* The logarithm of the DFT is written to file *file.da.*

m=*m.* The number of output points per row is set to *m.*

-c Takes the DFT of the columns of the input array.

-r Takes the DFT of the rows of the input array. This is the default mode of operation.

-p The program will not print out information and therefore runs silently.

SEE ALSO

qplot(1), **parse**(3), **parse_da**(3), **xpic**(1), **armi**(1), **asda**(1), **cda**(1), **dft**(1), **daas**(1), **dami**(1), **exda**(1), **fmm**(1), **minda**(1), **mag**(1)

AUTHOR

Carl R. Crawford

NAME

exda - extract data from da-files

SYNOPSIS

exda *[options] [file[.da]]*

DESCRIPTION

By default, **exda** extracts the first record from the *da*-file named *y.da* and writes it to a file named *y1.da.* The following options are available to change the default performance of the program:

if=*file[.da]*

file[.da] Data are extracted from *file[.da]* instead of the default file *y.da.* The suffix *da* is added if necessary.

of=*file[.da]* The extracted data are written to *file[.da]* instead of the default file *y1.da.* The suffix *da* is added if necessary.

-c Column mode is specified. By default when this option is used, the first elements in each record in the input file are collected to form a new *da*-file.

first=*r* The first record to be extracted is *r* instead of record zero, which is the default. In column mode, this specifies the first record from which to extract a specified point.

count=*n* The number of contiguous records to be extracted is specified by *n.* The default value is one. A synonym for this option is **n=.** In column mode, this specifies the number of records from which to extract a specified point.

start=*e* Data beginning with element *e* within each record are extracted. By default all the data in a record(s) are extracted. In column mode, this specifies which element in each record to extract.

end=*e* Data ending with element *e* within each record are extracted. By default all the data in a record(s) are extracted.

column=*c* specifies column mode, **-c,** and the element in each record to extract.

-P The user is prompted for program parameters.

-p The program will not print out information and therefore runs silently.

SEE ALSO

qplot(1), **parse**(3), **parse_da**(3), **xpic**(1), **armi**(1), **asda**(1), **cda**(1), **daas**(1), **dami**(1), **exda**(1), **fmm**(1), **minda**(1), **mag**(1)

AUTHOR

Carl R. Crawford

NAME

fmm - find minimum and maximum of data contained in da-files

SYNOPSIS

fmm *[-q]* file*[.da]...*

DESCRIPTION

fmm finds the minimum and maximum of data contained in *da*-files specified on the command line. These values are printed per file and also for the aggregate. The locations of the minima and maxima are printed in the form *(r,e),* where *r* is the record number and *e* is the element number within a record. The first record and element are numbered zero. The names of the files and their sizes (number of records x size of record) are also reported. See **parse_da**(3) for additional information on *da*-files.

If the **-q** flag is present, then the output will be formatted for use with **qplot**(1). Specifically, the output is in the form

ymin=*minimum* ymax=*maximum*

With this option the program can be used on the command line for **qplot**(1) to plot a number of vectors at the same scale. An example of this functionality is:

qplot y1.da ‘fmm -q y1.da y2.da y3.da‘

SEE ALSO

qplot(1), **parse**(3), **parse_da**(3), **xpic**(1), **armi**(1), **asda**(1), **cda**(1), **daas**(1), **dami**(1), **exda**(1), **fmm**(1), **minda**(1), **mag**(1)

AUTHOR

Carl R. Crawford

NAME

lc - extract lines and columns from xpic's canvas

SYNOPSIS

lc *[options]*

DESCRIPTION

lc reads rows (lines) and columns of data from a **xpic**(1) canvas. By default, the complete center row of the canvas is written out to a *da*-file named *line.da.* The *port* option in **xpic**(1) has to be enabled before the program will work. The port option is enabled either with the **-p** flag on **xpic**(1)'s command line or via the **port** command within **xpic**(1). Column and row numbers begin with zero and continue up to the total number minus one.

OPTIONS

The following options are available on the command line to modify the performance of the program:

-c Pick up data in a column instead of a row.

-l Pick up data in a row. This is the default mode.

-d Pick up row *or* column from the last deposited position of the cursor in **xpic**(1). Note that a cursor position in **xpic**(1) has to specified by depressing the left mouse button when the cursor is positioned over the canvas.

-D Pick up row *and* column from the last deposited position of the cursor in **xpic**(1). Note that a cursor position in **xpic**(1) has to specified by depressing the left mouse button when the cursor is positioned over the canvas.

-p Don't print out status of command.

-? Print out a help message. Note that the '?' has to be escaped (preceded with a '\') to hide it from the shell.

of=*file[.da]* The line or column is stored in the file named *file[.da].* The *.da* suffix is added if required.

x=*x* The starting point of a row is set to *x* if in row mode. In this mode, the default starting position is the first point (actually point zero) of the row. In column mode, the column is set to *x* .

y=*y* The starting point of a column is set to *y* if in column mode. In this mode, the default starting position is the first point (actually point zero) of the column. In row mode, the row is set to *y* .

n=*n* The number of points in a line or column is set to *n.* By default all the points from a line or column are read from the starting point.

host=*h* Extract data from the **xpic**(1) running on *h.* The default host is the machine on which the command is executed.

port=*p* Extract data via port number *p.* The default port is the 8125. This can used in conjunction with the **port=** command in **xpic**(1) to access multiple displays running on the same host.

SEE ALSO

cplot(1), **xpic**(1), **plot3d**(1), **qplot**(1), **xplot**(1), **dami**(1), **armi**(1), **crcplot**(3), **parse_mi**(3)

AUTHOR

Carl R. Crawford (ccrawford@analogic.com)

NAME

mag - magnify images contained in mi-files by two using bi-cubic interpolation

SYNOPSIS

mag *[-p]* infile*[.mi] [outfile[.mi]]*

DESCRIPTION

mag magnifies images contained in *mi*-files by a factor of two using bi-cubic interpolation. The image to be magnified is the first image contained in *infile.* The filename can also include the suffixes *:l* or *:L* to indicate that the last image should be read. The suffixes *:f,* or *:F* indicate that the first image should be read. The suffix *:n,* where *n* is an integer, indicates that image *n* should be read, where the first image is number one. If a suffix is used, the warning message is not printed. The resulting magnified image is either returned to *infile* or written to *outfile* if the latter file is specified. The suffix *mi* is added to the filenames if necessary.

The **-p** flag can be used to turn off the messages that the program normally displays.

SEE ALSO

parse(3), **parse_mi**(3), **xpic**(1), **armi**(1), **asda**(1), **cda**(1), **daas**(1), **dami**(1), **exda**(1), **fmm**(1), **minda**(1), **mag**(1)

AUTHOR

Paul Granfors wrote the magnification code and Carl R. Crawford surrounded it with a parser.

BUGS

If the input file contains multiple images and the output *outfile* is not specified, then the images not magnified in *infile* will be lost.

NAME

minda - minifies da-files

SYNOPSIS

minda *[options]*

DESCRIPTION

By default, **minda** minifies the data contained in a file *y.da* by a factor of two in each direction. Boxcar integration is used to lowpass filter the data before decimation. The resulting data is written to *min.da.*

The following options are available to change the default performance of the program:

if=*file[.da]* The input file is set to *file[.da]* instead of the default *y.da.* The suffix *da* is added if necessary.

of=*file[.da]* The output file is set to *file[.da]* instead of the default *min.da.* The suffix *da* is added if necessary.

xm=*x* The data are minified by a factor of *x* in the horizontal (x or column) direction. The default value is two.

ym=*y* The data are minified by a factor of *y* in the vertical (y or row) direction. The default value is two.

-t The program will transpose the array before output. The default is not to transpose the output.

-p The program will not print out information and therefore runs silently.

SEE ALSO

qplot(1), **parse**(3), **parse_da**(3), **xpic**(1), **armi**(1), **asda**(1), **cda**(1), **daas**(1), **dami**(1), **exda**(1), **fmm**(1), **mag**(1), **minda**(1)

AUTHOR

Chris Ruth (cruth@analogic.com)

NAME

plot3d – plot three dimensional surfaces

SYNOPSIS

plot3d [*options*] [*file[.da|.mi|.as]*]

DESCRIPTION

plot3d prints a view of a three dimensional surface. The resulting graphics is sent directly to the X-window, called a plotting window, maintained by **xplot**(1). **xplot**(1) has to be running before executing **plot3d**. **xplot**(1) and **plot3d** can be run on different hosts if the environment variable **XPLOTHOST** is used.

A matrix of binary floating point numbers in a da-file will be read from a file called *z.da* and used as the z value at each point. This vector will be displayed against a program generated xy-matrix containing integers from zero to the number of points in each direction minus one. The points in the file are assumed to be rows of values along lines of constant y.

The input file can be either a binary or an ASCII file. In a binary file the data is coded in big-endian format. The file can be generated using the **fwrite**(3s) statements in C or an unformatted write in Fortran or Pascal. This is the most efficient form since it saves on both file space and machine time. *mi*-files, containing images, and *da*-files, containing data with a header, can also be used as input. See **parse_da**(3) and **parse_mi**(3) for information on these file formats.

For simple applications it is also possible to use an ASCII file with the numbers in a readable format. The user must be careful to edit out any titles or other non-numeric information. The numbers are read from the file in "free format". In other words, spaces, tabs and newlines can all be used to separate the numbers. The program will read as many lines as necessary to get enough data for the plot as specified by the **count**, **skip** and **begin** options (see below for additional information). The numbers that are read from the file must not contain any spaces.

The program supports as many of the **qplot**(1) options as possible and in most cases the options/flags are known by the same name. The most significant difference is that the data file is indicated by the **z** option instead of the **y** option for obvious reasons.

The input data are a matrix of binary numbers. The type and location of the data is specified with the **z** option. The matrix is considered to be size 64 x 64 unless one of the size options (**size**, **xsize**, or **ysize**) is used. The use of *mi*- or *da*-files overrides this assumed size.

Data are read from the file along lines of constant y (x variable varies fastest). The first value in the file, file(0), is z(x=0, y=0), the second is z(x=1, y=0) and so on. The second line (y=1) starts at number **xsize** in the file (file(**xsize**) --> z(x=0, y=1)). It is very important that the variable **xsize** be set properly so that the program can index into the matrix properly.

With the **x** and **y** options an arbitrary mapping can be made between points in the matrix (z) and the x and y axis. Normally each line of the data is assumed to be equally spaced in x and y. With the **x** and **y** options any arbitrary mapping is possible. The results are guaranteed to be meaningful only for monotonically increasing mappings.

With the **begin**, **skip** and **count** parameters it is possible to specify that part of the input data be ignored. If the **begin** options are used, the first **xbegin** data points of the z (per row) and x files are ignored. A similar effect is seen if the **ybegin** parameter is specified. The default values are 0. If the **skip** options (**skip**, **xskip**, and **yskip**) are set then lines in the input matrix are skipped between each point that is plotted. The **count** option is used to set the number of data lines to plot. Normally the **count** parameters default to the largest number possible, given the **size**, **begin** and **skip** variables.

OPTIONS

The following options and flags are available on the command line to modify the performance of the program. Options can be minimally abbreviated.

****** FILE SPECIFICATION OPTIONS ******

z=*file[,n]* The z matrix will be read from *file* instead of *z*. The letter *n* is the data type declaration field and can be one of the following:

- **c** Single bytes, unsigned, fixed precision **char** data.
- **cs** Single bytes, signed, fixed precision **char** data.
- **s** Two bytes, signed, fixed precision short **int** data.
- **i** Four bytes, signed, fixed precision **int** data.
- **l** Four bytes, signed, fixed precision **long int** data.
- **f** Four bytes, **float** data.
- **p** Eight bytes, **double** floating-point data.
- **a** ASCII data, free format numbers, data is readable, spaces, tabs and newlines are used to separate input points.
- **d** *da*-file, see the appendix of **parse_da**(3) for file format.
- **m** *mi*-file, see the appendix of **parse_mi**(3) for file format.

If *n* isn't specified, a comma is not needed after the file name. The default for *n* is **d**.

z=,*n* The z vector is read from the default file *z* but the byte declaration field is set to *n*.

-*x* Read the x vector from the file *x*. The data in the file is assumed to be in a da-file.

x=*file[,n]* The x vector will be read from the file *file* instead of *x.da*. The data in the file is assumed to be a da-file unless a *,n* is added. The variable *n* can be any of the suffixes shown above with the **z** option.

-**y** Read the y vector from the file *y*. The data in the file is assumed to be in a da-file.

y=*file[,n]* The y vector will be read from the file *file* instead of *y.da*. The data in the file is assumed to be in a da-file. unless a *,n* is added. The value of *n* can be any of the suffixes shown above with the **z** option.

xsize=*n* The matrix has *n* elements in the x-direction. The default value is 64. It is necessary to specify this variable if the array is not 64 x 64. The program interprets the z-file as an **xsize** by **ysize** matrix of values. It is not necessary to specify **xsize** or **ysize** if the input format is *da*-files or *mi*-files.

ysize=*n* The matrix has *n* elements in the y-direction. The default value is 64. The program interprets the z-file as an **xsize** by **ysize** matrix of values. It is not necessary to specify **xsize** or **ysize** if the input format is *da*-files or *mi*-files.

size=*n* Set both **xsize** and **ysize** equal to *n*.

n=*n* This sets the **xsize** and **ysize** variables equal to *n*. This is equivalent to using the **size** option.

****** DATA SELECTION OPTIONS ******

xbegin=*n* The first *n* columns of the input in the z-file (per row) and *n* values in the x-file are skipped. This is used to place the origin of the plot at an arbitrary position in the matrix. The default value is 0.

ybegin=*n* The first *n* rows of the input in the z-file and *n* values in the y=file are skipped. This is used to place the origin of the plot at an arbitrary position in the matrix. The default value is 0.

begin=*n* Set the parameters **xbegin** and **ybegin** equal to *n*.

xskip=*n* Skip *n* columns in the z-file (per row) and *n* values in the x-file between plotted data points. The default value is 0.

yskip=*n* Skip *n* rows in the z-file and *n* values in the y-file between plotted data points. The default value is 0.

skip=*n* Set the parameters **xskip** and **yskip** equal to *n*.

xcount=*n* Only plot *n* of the columns in the matrix. Show *n* values in the matrix for each line of constant y. The default value is the largest number of points that can be plotted for the given **xsize**, **xbegin** and **xskip**.

ycount=*n* Only plot *n* of the rows in the matrix. Show *n* values in the matrix for each line of constant x. The default value is the largest number of points that can be plotted for the given **ysize**, **ybegin** and **yskip.**

count=*n* Set the parameters **xcount** and **ycount** equal to *n*.

zmin=*r* All points in the matrix below the level *r* are set to *r*. This is useful for seeing details in the surface. See also the **zmax** option. The default value is obtained from the data itself.

zmax=*r* All points in the matrix above the level *r* are set to *r*. This is useful for seeing details in the surface. See also the **zmin** parameter. The default value is obtained from the data itself.

xmin=*min* Normally the x-axis is annotated with values from zero to the number of y lines minus one. With this option the minimum value on the x-axis is set to *min.*

xmax=*max* Only if used with the *xmin* option, floating-point annotations will be used between the value set with this option, *max*, and the value set with the **xmin** option.

ymin=*min* Normally the y-axis is annotated with values from zero to the number of x lines minus one. With this option the minimum value on the y-axis is set to *min.*

ymax=*max* Only if used with the **ymin** option, floating-point annotations will be used between the value set with this option, *max*, and the value set with the **ymin** option.

****** PLOT POSITIONING AND REDUCING OPTIONS ******

xp=*r* The x starting coordinate of the axes is moved to position *r* on the plot plane. The default value of *r* is 0. The plot is 10 units wide.

yp=*r* Same as the **xp** option but the y origin is moved to *r* on the plot plane. The plot is 10 units high.

scfac=*r* The graph is expanded or reduced in size by *r*. The default value is 1.0. The option **scale** is a synonym for this parameter. There is another scale factor of two built into the program.

-e Do not erase the **xplot**(1) window before plotting. This is useful for overlaying multiple vectors on the same set of axes.

-o Send the **plot**(5) format output to **stdout** instead of via a socket connection to **xplot**(1).

host=*[host][:socket]*
Connect to the plotting window on the machine named *host* using socket number *socket*. If the host is not specified, then the plotting window on the local host will be used. If the host is the string *stdout* the **plot**(5) commands will be sent to standard output. If the socket is not specified, then the socket listed in the section INTERPROCESS COMMUNICATION will be used.

****** AXIS OPTIONS ******

-a No axes will be plotted.

axis=*[xyz]* An axis will be plotted only for axis listed after the equals sign. The default action is *xyz* which labels all three axis of the plot.

-f A border will be drawn around the plot.

len=*r* Set the length of all axes to *r*.

xlen=*r* The length of the x axis is changed from eight plot units to *r* units. This parameter is relative to the default value of 6. The actual length of the x axis in the 2d plot coordinate system is a function of the angle of view and the **scfac**.

ylen=*r* The length of the y axis is changed from eight plot units to *r* units. This parameter is relative to the default value of 6. The actual length of the y axis in the 2d plot coordinate system is a function of the angle of view and the **scfac**.

zlen=*r* The length of the z axis is changed to *r* units. This parameter is relative to the default value of four. The actual length of the y axis in the 2d plot coordinate system is a function of the angle of view and the **scfac**.

****** LABEL OPTIONS ******

tl=*str* The string *str* will be used as a label at the top of the graph.

bl=*str* The string *str* will be used as a label at the bottom of the graph.

xl=*str* The string *str* is used as a label along the x axis.

yl=*str* The string *str* is used as a label along the y axis.

zl=*str* The string *str* is used as a label next the z axis.

****** GRAPH OPTIONS ******

phi1=*theta* The surface is rotated around the z-axis by an angle of *theta* before plotting. The zero angle corresponds to looking down the y axis. The default value is 40 degrees. **rz** is a synonym.

phi2=*theta* The observation plane is tilted by *theta* degrees about the horizontal axis. Zero degrees corresponds to looking at the surface from the xy-plane; 90 degrees is looking at the object from directly above. Neither of the two views at the extremes provide much information. The default value is 30 degrees. **rx** is a synonym.

-r Reverse the direction of the z-axis. This effectively multiplies each data point by -1.

direction=*[xy]* Plot lines along the axis (or axes) specified by this parameter. The default is to plot the lines along the y axis. The graph will often appear simpler if lines are only drawn in one direction.

resolution=*x* This parameter controls the resolution of the hidden surface removal subroutine. The default value is 1.0 and larger values can be used to produce a more accurate estimate of the line intersections at the expense of more computer time. Values larger than one are generally needed only for publication quality plots of functions with large number of discontinuities. Conversely a value smaller than one will save computer time at the expense of small errors in the intersections. The valid range for *x* is (0.7,4.0).

DEFAULTS

The internal parse routines allow for two default mechanisms to specify options. The first method is to create files **.plot3d** in your **HOME** directory and/or **.plot3d** in the working directory. In them, one can place options and flags that will be used before parsing the command line. Lines beginning with a '#' are considered to be comments. A '#' also marks the beginning of a comment anywhere else on an input line. Parsing of arguments ends when either an end-of-file is reached or when the line beginning with the string "//" is found. The second method to set default options is to use the environment variable **PLOT3DARGS.** The format is the same as the input command line. The order or parsing is: **~/.plot3d ./.plot3d, PLOT3DARGS,** and finally the command line. The flag **-@,** if present on the command line, will disable the use of the default mechanisms.

FILES

~/.plot3d

./.plot3d initialization (startup) files.

ENVIRONMENT

PLOT3DARGS contains command line arguments.

HOME contains the shell's concept of your home directory.

XPLOTHOST Can be used to specify the host and the socket for the plotting window. They have the form *[host][:socket].* The first one is used if both are set.

INTERPROCESS COMMUNICATION

8124 Default socket used to talk to plotting window.

SEE ALSO

qplot(1), **crcplot**(3), **plot3d**(3), **symbol**(3), **scale**(3), **parse_da**(3), **parse_mi**(3), **parse**(3), **parse_parse**(3), **parse_canonical**(3), **plot**(5), **cplot**(1), **plotps**(1), **xpic**(1), **xplot**(1)

AUTHORS

Mani Azimi assembled the basic hidden line removal routines that **plot3d** is based upon from software written by a couple of people. He also converted the code from Fortran to C and fixed a number of bugs. Malcolm Slaney combined Azimi's code with Carl Crawford's original version of **qplot.** Crawford adapted Slaney's version for use at GE Medical Systems and added the *da-* and *mi-*file formats. Funding for the development of this software was originally provided by Professor Avi Kak of the Electrical Engineering Department at Purdue University.

BUGS

No check is made to insure that the number of points to read from the input file for each line (**count***(**skip**+1)+**begin**) is consistent with the **size** options.

The resolution parameter really shouldn't be necessary.

Plots at angles near multiples of ninety degrees have some problems.

NAME

plotps – convert plot(5) files to Postscript

SYNOPSIS

plotps *[options] [file]*

DESCRIPTION

plotps reads **plot**(5) format commands from standard input and converts them to POSTSCRIPT format on the standard output. The conversion is almost one-for-one, with one POSTSCRIPT function call for each *plot* primitive. By default, the output is printed at 60% of full scale. Usually the program is piped into **lpr**(1); in this case the **-g** flag for **lpr**(1) should not be used.

OPTIONS

The following options are available on the command line to modify the performance of the program:

scale=*s* Change the percent of full size of the output, where the units on this option are [1,100]. The default is 60%.

-e Encapsulated postscript (EPS) is generated. EPS an can be used in Interleaf, Ditroff and in TeX.

-x Same as the -e option with the exception that the plot commands are taken from the current display of **xplot**(1).

-X The plot commands are taken from the current display of **xplot**(1).

-l Print in the landscape orientation.

-p Print in the portrait orientation. This is the default orientation.

hor=*h* The horizontal offset of the plot is changed from 0.5 to *h* inches. The offset is ignored when EPS is generated.

ver=*v* The vertical offset of the plot is changed from 0.5 to *v* inches. The offset is ignored when EPS is generated.

bbx=*sx*

bby=*sy* The horizontal and vertical sizes of the bounding box used for EPS are scaled by *sx* and *sy,* respectively. Their default values are one. These parameters are used to shrink or grow the bounding box with respect to the actual plotting area defined with the **space**(3) or **plot_space**(3) (part of **plotcrc**(3)) commands. These commands invoke the **-e** option.

SEE ALSO

xplot(1), **qplot**(1), **plot3d**(1), **cplot**(1), **crcplot**(3), **plotcrc**(3), **plot**(3x), **plot**(5), **space**(3), **lpr**(1)

AUTHOR

Carl R. Crawford

BUGS

A way is needed to change the position, orientation, and absolute size of the final output.

In EPS mode, there is no check to make sure that bounding box really bounds the graphics.

NOTES

POSTSCRIPT is a registered trademark of Adobe Systems Incorporated.

NAME

qplot – quickly plot vectors

SYNOPSIS

qplot [*options*] [*file[.da|.as]*]

DESCRIPTION

qplot takes as its input a y-vector and optionally an x-vector and produces as its output an x-versus-y plot of the input data. The resulting graphics is sent directly to the X-window, called a plotting window, maintained by **xplot**(1). **xplot**(1) has to be running before executing **qplot**. **xplot**(1) and **qplot** can be run on different hosts if the environment variable **XPLOTHOST** is used.

The program is basically a parser built around a plotting core. The parser provides options to control the relatively simple graphics output. The idea behind the program is that in scientific applications the first priority is to provide a method to visualize data without having to write a large amount of software for each new application. The program provides a method to supply the user with simple graphics output for many applications.

The default input file is *y.da* and is assumed to be in *da*-file format. See the appendix of **parse_da**(3) for a description of *da*-files. The input files can also be in ASCII format. In this case, the data consists of up to 200 columns of numbers. The numbers should be separated with spaces or tabs. A number itself cannot contain any spaces. The user must be careful to edit out any titles or other non-numeric information before plotting. The program also supports raw binary files that do not contain any file headers.

OPTIONS

The following options and flags are available on the command line to modify the performance of the program. Options can be minimally abbreviated.

****** FILE SPECIFICATION OPTIONS ******

y=*file*

file — The y vector is read from *file*. The default file-type suffix is *.da*. If the **-c** option is used then the default is *.as*. If the **-b** flag is used, then the file is assumed to be in raw binary format and no suffix is added. For binary files, the **type** option can be used to change the default format of the binary file.

x=*file* — Read the x-vector from *file*. The default file-type suffix is *.da* or *.as* if the **-c** flag is used.

-x — Equivalent to **x=x.da** or **x=x.as** if the **-c** flag is invoked.

-c — Invoke the character mode. In this mode data is read from *as*-files instead of *da*-files. An *as*-file is an ASCII file containing columns of numbers.

cxy=*file* — The **-c** flag is invoked and the x- and y-vectors are extracted from the first two columns of *file*. The default file name is cxy[.as].

nx=*col* — The x-vector is read from column *col* instead of column one. If the **cxy** option has not been used, then the **-x** flag will be invoked.

ny=*col* — The y-vector will be read from column *col* instead of column one. If the **cxy** option has not been used, then the column number is absolute. If the **cxy** option has been used, then the column number is relative to the x-column number and *col* can be negative.

-b — Specifies that the input files are raw binary instead of *da*-format or ASCII.

type=*t* — Set the default binary file type to *t*, where *t* is in the set {r,f,i,l,d,s} which corresponds to real, float, integer, long, double and short, respectively. The default type is float. The data have to be in big-endian format.

****** DATA SELECTION OPTIONS ******

count=*c* Read only *c* points from the input vectors. If *c* is negative, then the units on *c* are records. The default value of *c* is -1.

skip=*n* Read only every *n*'th plus 1 point from the input vectors. The default value is zero.

begin=*b* Begin plotting with the *b*'th point in the input vectors. The first point in a vector is *b*=0. If *b* is negative, then the units on *b* are records.

ymax=*max* Normally the maximum value on the y-axis is the maximum value found in the input vector. This option overrides this feature and sets the value on the axis to *max.*

ymin=*min* Same as the **ymax** option but works on the minimum value for the y-axis.

xmin=*min* Normally the x-axis is annotated with values from zero to the number of points in the y vector minus one. Note that there is no maximum number of points. With this option the minimum value on the x-axis is set to *min.*

xmax=*max* Only if used with the **xmin** option, floating-point annotations will be used between the value set with this option, *max*, and the value set with the **xmin** option.

-**n** Override the minimum and maximum values on the axes and put in "nice" values using **scale**(3).

-**s** Save the XMIN, XMAX, YMIN, and YMAX values used on the axes in a file named *xy.as.* The format of the file is (XMIN, XMAX, YMIN, YMAX) written in e-format. The values can be used by another execution of the program using the **s** option.

s=*file* There are two modes for this option. If this option appears with the -**s** flag, then the actions dictated by the use of -**s** will be followed with the exception that the scale information will be saved in the file *file[.as]* . If this option appears without -**s**, then the values in *file[.as]* will be read in and used as scale values for the current plot. The **xmin**, **xmax**, **ymin** and **ymax** options can be used to override the values contained in the scale file.

****** PLOT POSITIONING AND LOOK, OPTIONS ******

scale=*s*

scfac=*s* Scale the final output by *s.* The default value is one.

xp=*x* Move the complete graphics output by *x* plot-units along the x-axis. The value can be positive or negative.

yp=*y* Move the complete graphics output by *y* plot-units along the y-axis. The value can be positive or negative.

-**f** Draw a frame around the graph.

-**e** Don't erase the display before plotting.

-**g** Superimpose a grid over the plot. The grid connects the tic-marks on the axes to each other through a set of orthogonal lines.

-o Send the **plot**(5) format output to **stdout** instead of via a socket connection to **xplot**(1).

host=*[host][:socket]*
Connect to the plotting window on the machine named *host* using socket number *socket.* If the host is not specified, then the plotting window on the local host will be

used. If the host is the string *stdout* the **plot**(5) commands will be sent to standard output. If the socket is not specified, then the socket listed in the section INTERPROCESS COMMUNICATION will be used.

quadrant=*q*

nanant=*n*

hexant=*h* The output window is broken up into 4, 9 or 16 regions for the *quadrant, nanant,* and *hexant* commands, respectively. The graphics output is scaled and translated to fit into the specified region. If the region is negative, then the subregion will be shifted over to the right by the field-of-view.

****** AXIS OPTIONS ******

-**a** Don't plot the axes.

xlen=*l* Set the length of the x-axis to *l* plot-units. The default length is three plot-units.

ylen=*l* Set the length of the y-axis to *l* plot-units. The default length is three plot-units.

len=*l* Set the length of both axes to *l* plot-units.

****** GRAPH OPTIONS ******

-**d** Plot the vector with a dashed line. The length of the dash and gap segments of the dashed line are both 0.1 plot units by default.

dash=*d* Set the length of the dash section of a dashed line to *d* plot-units. This option invokes the -**d** flag.

gap=*g* Set the length of the gap section of a dashed line to *g* plot-units. This option invokes the -**d** flag.

-**m** Mark every point in the graph of the vector with an on-line-symbol.

j=*n* Invoke the -**m** flag, but mark only every *n*'th point. If *n* is negative, do not connect the symbols with lines.

sym=*s* Invoke the -**m** flag, but use the *s*'th symbol for marking purposes. The descriptions of the symbols can be found in **symbol**(3).

****** LABEL OPTIONS ******

xl=*label* Plot the string *label* below the x-axis. Double quotation marks must be used if the label contains spaces or meta-characters.

yl=*label* Plot *label* next to the y-axis. Double quotation marks must be used if the label contains spaces or meta-characters.

tl=*label* Plot the string *label* above the graph. Double quotation marks must be used if the label contains spaces or meta-characters.

bl=*label* Plot the string *label* below the x-axis label. Double quotation marks must be used if the label contains spaces or meta-characters.

el=*label* The string *label* will be plotted just to the right of the last point of the plotted vector.

-l The user will be prompted via the terminal for labels not entered on the command line. A null label can be entered by entering <nl> after a prompt.

-L The file names used for the x- and y-vectors are used as labels for the axes.

DEFAULTS

The internal parse routines allow for two default mechanisms to specify options. The first method is to create files **.qplot** in your **HOME** directory and/or **.qplot** in the working directory. In them, one can place options and flags that will be used before parsing the command line. Lines beginning with a '#' are considered to be comments. A '#' also marks the beginning of a comment anywhere else on an input line. Parsing of arguments ends when either an end-of-file is reached or when the line beginning with the string "//" is found. The second method to set default options is to use the environment variable **QPLOTARGS.** The format is the same as the input command line. The order or parsing is: **~/.qplot ./.qplot, QPLOTARGS,** and finally the command line. The flag **-@,** if present on the command line, will disable the use of the default mechanisms.

FILES

~/.qplot

./.qplot initialization (startup) files.

ENVIRONMENT

QPLOTARGS contains command line arguments.

HOME contains the shell's concept of your home directory.

XPLOTHOST Can be used to specify the host and the socket for the plotting window. It has the form *[host][:socket].* The first one is used if both are set.

INTERPROCESS COMMUNICATION

8124 Default socket used to talk to plotting window.

SEE ALSO

cplot(1), **crcplot**(3), **symbol**(3), **scale**(3), **parse_da**(3), **parse**(3), **parse_parse**(3), **parse_canonical**(3), **plot**(5), **plot3d**(1), **plotps**(1), **xpic**(1), **xplot**(1)

AUTHOR

Carl R. Crawford

BUGS

The program tends to hang will hang if invalid floating point data is contained in the input vectors.

NAME

sagcor - generate sagittal, coronal and and projection images from mi-files

SYNOPSIS

sagcor *[options] [file[.mi]]*

DESCRIPTION

By default, **sagcor** generates the central coronal slice from the images contained in the *mi*-file named *pic.mi* and writes the resulting coronal slice to a file named *sc.mi.* The program can also generate sagittal and projection images. The following options are available to change the default performance of the program:

if=*file[.mi]*

file[.mi] Images are extracted from *file[.mi]* instead of the default file *pic.mi.* The suffix *mi* is added if necessary.

-v Generates a synthetic view of the resulting coronal or sagittal slices. That is, the coronal or sagittal slices are generated as per above. Then the set of slices is averaged to generate one output slice.

of=*file[.mi]* The generated images are written to *file[.mi]* instead of the default file *sc.mi.* The suffix *mi* is added if necessary. Synthetic views are written with *parse_wmi* so that "__" can be used to append files. Otherwise, the mi-file is written from scratch.

-c Generates coronal slices. This is the default image type.

-s Generates sagittal slices. The default image type is coronal.

fimage=*first*

limage=*last* The program by default uses all the images in the input file. These options restrict the program to using images from *first* to *last.* The first image is image one.

row=*row* Generates the coronal image at row *row* instead of the central row. Row one is the top row of the input images.

col=*col* Generates the sagittal image at column *col* instead of the central column. Column one is the left column of the input images.

n=*n* Generates *n* images beginning at *row* or *col.*

-a Generates *all* of the possible coronal or sagittal images. This effectively sets *n* to the number of rows or columns in the input images.

-p The program will not print out information and therefore runs silently.

offset=*offset* The number *offset* is subtracted from every input pixel. The result is set to zero if the subtraction is less than zero.

SEE ALSO

crc(1)

AUTHOR

Carl R. Crawford

NAME

xpc - control xplot

SYNOPSIS

xpc
xpc -h
xpc -p
xpc -a *text...*

DESCRIPTION

xpc is used to control some of the functions of **xplot**(1).

When no options are specified, the program clears the **xplot**(1) plotting area. This is equivalent to using the **erase**(3) function in **crcplot**(3) and to using the **Erase** button in **xplot**(1).

When **-h** is present, the program sends the graphics contained in the plotting area of **xplot**(1) to the line-printer. This is equivalent to using the **hardcopy**(3) function in **crcplot**(3) and to using the **Hardcopy** button in **xplot**(1).

When the **-a** flag is specified, the rest of the command line is used to annotate the bottom of the plotting area in **xplot**(1). The current date and time are placed before the user-specified text.

When the **-p** option is used, **stdin** is copied to **xplot**(1). This mode is used to send **plot**(5) commands to **xplot**(1) when the originating program does not use **plotcrc**(3).

ENVIRONMENT

XPLOTHOST the host on which **xplot**(1) is running. If not specified, **xplot**(1) on the current host is used. If the name of the host is **stdout**, the plotting commands in **plot**(5) format will be sent to standard output.

SEE ALSO

cplot(1), **qplot**(1), **crcplot**(3), **symbol**(3), **scale**(3), **parse_da**(3), **parse_intro**(3), **parse_parse**(3), **plot**(5), **plot3d**(1), **plotps**(1), **xpic**(1), **xplot**(1)

AUTHOR

Carl R. Crawford

NAME

xpic – display images in the X environment

SYNOPSIS

xpic [*generic-tool-args*] [*xpic-options*] [*xpic-flags*]

DESCRIPTION

xpic consists of four windows: a canvas in which images are displayed; a tty in which user commands are entered; a panel in which some commands can be invoked and messages are displayed; and a graphical file browser which can optionally be displayed. The default sizes of the canvas and the tty windows can be altered on the command line. By default xpic's browser is hidden. Provision has been made for subdividing the canvas into subcanvases in which different images can be displayed.

xpic reads and displays images in either *mi*-file or LM format. Both the original and and full field-of-view versions are supported at compile time. In the case of LM files, the high or low axial images, or the 0, 45, or 90 degree projections can be displayed. The program bounces the current canvas definition against the size of the image, and does the best it can to display the image. If the image to be read is smaller than the current canvas, it will be written to the center of the canvas. If larger, the center portion fitting the canvas will be displayed.

The basic operating mode of **xpic** is as follows. The user selects a default operating directory using *cd.* Then a file containing an image is opened using **open** or the file browser. Finally, the images in the file can be displayed by specifying the actual image numbers.

COMMANDS

The following commands can be entered in the tty window. The string **xpic>** is used as a prompt to indicate that xpic is ready for input. All commands can be abbreviated to their shortest unique prefix. Information contained between square brackets ([]), is optional. A list of choices is specified by having the elements of the list separated by vertical bars (|). Sometimes the list will be surrounded by curly brackets ({}).

**** FILE AND IMAGE DISPLAY COMMANDS ****

cd [*dir*] Changes the current working directory to *dir.* The default working directory is the directory from which **xpic** was invoked. If no arguments are specified, then the working directory will be changed to the value specified in the environmental variable HOME.

open *[~[user]][/dir1/.../dirn]file[{.mi|.mi.gz|.lm}][={i|z|l[hlp]}] ...*

Open accepts n arguments of the specified format. Opens a file named *file* from the home directory of the user *user* or the current user if none is specified. The expression *file* may contain shell style wildcards, * *?* *[* *]* ^. The * wildcard specifies any string of characters. This includes a NULL string. The *?* wildcard specifies any single character. The *[* and *]* characters can be used to indicate a range of valid characters. Finally, the ^ character, used in conjunction with the square brackets indicates a range of characters to exclude in a search, rather than include. A directory path, *dir1* through *dirn* may be specified. The actual file opened is relative to the current working directory. See the **cd** command for additional information. The *=i, =l,* and *=z* suffixes indicate the the file type is a *MI, LM,* and *GZMI* respectively. The *GZMI* file type indicates a gzipped *MI* file, with the suffix *.mi.gz.* In the case of LM files, the high or low axial images, or the projections can be displayed using the additional suffixes of *h, l,* and *p,* respectively. If no additional suffix is specified for an *LM* file, the default is *h.* Image numbers 1, 2, and 3, correspond to the 0, 45 and 90 degree projections respectively. The default image type for LM files is high energy axial images. The file type remains set until a different suffix is specified. The suffix *.mi* is appended to the *file* if the file type is set to *MI.* The suffix *.lm* is appended to the *file* if the file type is set to *LM.* The

suffix *.mi.gz* is appended to the *file* if the file type is set to *MIGZ*. If this command is invoked with the suffix and without a file name, then the file type will be set to the specified suffix. The *reset* command is automatically called if the *autoreset* option is set via *set*.

#
A command which is a single number will cause that image to be written into the current canvas or subcanvas if a subcanvas has been defined. The subcanvas is automatically cleared if the *autoclear* option is set via *set*. The automatic window/level function of the *wl* command is automatically performed if the *autowl* option is set via *set*.

#,#/#...
A list of numbers, possibly with one or more slash characters ('/') included will command the program to put up the listed images in a multiple-image mode. Image numbers may be separated by spaces or commas. A slash ('/') denotes the end of a row of image numbers to be written horizontally. An implied '/' character is assumed at the end of the list, so don't put one there unless you want an empty row. The list is evaluated, and the canvas subdivided as follows: the vertical size of each small image is the full current canvas height divided by the number of rows implied by the list (the number of '/' characters plus one). Within each row, the width of each small image is the full canvas width divided by the number of images in that row. Any subcanvas definition is canceled when this mode is invoked.

Examples of multiple-image display mode:

"1-3/4-6/9-7" would result in a 3 x 3 display; images 1,2,3 on top, images 4,5,6 in center row, and 9,8,7 on bottom.

"5/7/8-10" would give image 5 across the top, 7 in middle, and 8,9,10 across the bottom.

"1,2,3//4 5 6/" would give 1,2,3 across the top, an empty row, then 4,5,6, and another empty row (remember the implied '/' at the end). After all is done, you could go to a different file and do multiple display of "/1 2 3//4 5 6" to sandwich the corresponding images in between the others.

#,#|#...
Same as the "#,#/#..." command with the exception that '|' specifies a column of images instead of a row of images. A specification of image numbers cannot include both row ('/') and column ('|') characters.

#,#,#-#@
Same as the "#,#/#..." command with the exception that the images are displayed in a grid that is determined by the number of images specified. The image buffer is cleared before the images are displayed.

images [*list*]
Executes previous set of commands as if *images* was not typed. The purpose of this command is to support the *help* command for image loading.

+
The next image in the file is displayed in the current subcanvas. If the current image is the last image in the file, then the first image is displayed. If a file has been opened but no images have been previously displayed, then the first image is displayed. The button labeled "+" on the panel performs the same function as this command. The increment between images is the value *skip* set with the *set* command; the default value is one.

-
The previous image in the file is displayed in the current subcanvas. If the current image is the first image in the file, then the last image is displayed. If a file has been opened but no images have been previously displayed, then the first image is displayed. The increment between images is the negative of the value of *skip* set with the *set* command; the default value is negative one. The button labeled "-" on the panel performs the same function as this command.

file *[root:suffix:]file1,file2/file3...*
Same as the "#,#/#..." "#,#|#..." commands with the exception that each tile is filled with the first image in the specified file names. If *root* and/or *suffix* are present, they are prepended and appended, respectively, to the rest of the file names.

cine *[first last [pause] [-s skip]]*

Displays the images from *first* to *last* in sequential order. The command is equivalent to sending *first, first+1, ...* The command pauses between images by an amount equal to *pause* milliseconds, where the default is zero milliseconds. The command remembers previous setting of *first, last,* and *pause* so that the command can be repeated when no arguments are present. The command traverses images from greatest to least if *first* is larger than *last.* The *-s* switch may be added to the end of the a *cine* command to specify the number of frames xpic increments by between screen updates. This option can be used regardless of whether *pause* is specified. The *skip* argument is the number of frames xpic will increment or decrement by each time it changes an image. *Skip* should be a positive integer. When traversing in the positive direction (incrementing image indices) through an image file the final image will be the greatest integer (including zero) multiple of *skip* less than the specified final image, offset by the first image specified. When traversing in the negative direction (decrementing image indices), the inverse rule applies.

reopen *[type]* The file previously opened with the **open** command is closed and then reopened. This command is useful when an external program has changed the contents of the file. If *type* is present, then *=type* is appended to the filename before reopening the file. This mode can be used to change the type of image read from a file. At present, it is only useful with the LM file format for changing between high energy, low energy, and projection images.

browse [*path*] Open the graphical browser window to display the contents of *path.* If no path is specified, then the browser will open to display the contents of the current working directory. **difference** [*image* [*offset*]] Subtract image numbered *image* from the image currently loaded in the canvas. By default, 1024 will be added to the difference. Setting *offset* will change the additive offset. The offset stays in affect until the next time *offset* is specified. The command without any arguments prints the value of the offset.

center *x y* Selects the center coordinate, *(x,y),* of the input image(s) to be placed in the center of the display canvas (or subcanvas). Coordinates are (0,0) for the upper left-hand pixel of an image, and increase in x to the right, and y to the bottom. Entering this command without any arguments will revert to the default mode which places the image's center at the current canvas's center.

corner *x y* Selects the coordinate, *(x,y),* of the input image(s) to be placed in the upper-left corner of the display canvas (or subcanvas). Coordinates are (0,0) for the upper left-hand pixel of an image, and increase in x to the right, and y to the bottom. Entering this command without any arguments will revert to the default mode which places the image's center at the current canvas's center.

scroll *x y* Moves the center or corner set with the *center* or *corner* commands by *(x,y)* and redisplays the images. If neither the center nor the corner are set, then the center is set to the center of the image and then scrolled. The new values of the center or the corner are calculated modulo the x- and y-extents of the image. If *x,y* are not specified, then the previous values of the command are used. The default values of *x* and *y* are the x- and y-sizes of the canvas, respectively. The values of *x* and *y* can also be set using the *set* command using the variables *scrollx* and *scrolly.* The canvas can also be scrolled using the panel buttons "u", "d", "l" and "r"; see the Control Panel section for additional information. When scrolling, the *autoclear* function can be used to prevent the canvas from being littered with old versions of the image.

environment [{[*[-d] env*]|+|-}]

Without arguments, the commands lists the ten environments supported by *xpic.* A file can be opened in each environment and each each environment can be a different directory and different file type. The directory, image name, and image type are printed for each environment. The current environment is indicated with an asterisk. If *env* is

specified (in the range one to ten), then the environment is changed to the specified number. If *-d* is present, then the current environment is duplicated in the designated environment; if a file was open in the new environment, it is closed; if a file was open in the current environment, it is not left open in the new environment. If + is specified, the environment is incremented to the next one, unless current environment is the maximum environment. In this case the current environment is set to the lowest possible environment. If - is specified, the environment is decremented to the previous one, unless current environment is the minimum environment. In this case the current environment is set to the highest possible environment.

****** SUBCANVAS COMMANDS ******

quadrant *q* Request that images be displayed in a quadrant *q* of the original canvas. The valid range for *q* is [1,4]. If *q* is + or - then the next or prior quadrant will be referenced.

nanant *n* Request that images be displayed in a *nanant n* of the original canvas. The valid range for *n* is [1,9]. If *n* is + or - then the next or prior nanant will be referenced.

hexant *h* Request that images be displayed in a *hexant h* of the original canvas. The valid range for *h* is [1,16]. If *h* is + or - then the next or prior hexant will be referenced.

horizontal *o* [*d*] Set the horizontal offset and dimension from the left boundary of the canvas to *o* and *d,* respectively. The default dimension is the current horizontal size of the canvas or subcanvas.

vertical *o* [*d*] Set the vertical offset and dimension from the top boundary of the canvas to *o* and *d,* respectively. The default dimension is the current vertical size of the canvas or subcanvas.

reset Resets any subcanvas specification. The default states of the center and corner commands are also set with this command.

copyquad *s d* Copy the contents of the source quadrant, *s,* to the destination quadrant, *d.* The subcanvas is set to the destination quadrant.

****** CURSOR AND ANNOTATION COMMANDS ******

cursor *[on | off]*

cursor *[-d]*

cursor -p *x y* Turns on or off cursor reporting. If no arguments are specified, then the status of the reporting function is toggled. When the mouse is positioned inside the canvas, the x and y locations of the mouse along with the value of the image are reported in the lower left corner of the canvas or in the panel. If the box cursor is enabled (see below), then only the location of the center of the box will be reported. If the left mouse button is depressed, then the position of the cursor and the value of the image at that location will be displayed in the tty window. The *-d* flag indicates that the cursor should be deposited. Depressing the right mouse button will also deposit the cursor. The *-p* flag positions the cursor at the specified x- and y-location. The value of the image at that location is also printed.

box *[on | off]*

box -s *x y*

box -p *x y* Turns on or off the box cursor capability. If no arguments are specified, then the status of the reporting function is toggled. The command invokes **cursor** if necessary. The center of the box is reported in the lower left hand corner of the canvas or in the panel. If the left mouse button is depressed, then the mean and standard deviation of the ROI defined by the box are reported in the lower left hand corner of the canvas or in the panel. The size of the box can be varied by dragging the mouse with the middle button depressed. The mean and standard deviation, along with the minimum and maximum pixels in the box, are also reported in the tty window. On the line below the mean, the center and size of the box are also reported. If the -**s** flag is present, then the size of the box will be set to *x* by *y*. If the -**p** flag is present, then the position of the box will be set to *x* and *y*. The box will also be enabled with the **-s** and the **-p** flags.

roi *[-e|b] [x y [xw yw]]*
Calculates the statistics of an image in a region-of-interest (ROI). The default region is the presently displayed box cursor. See the **box** command for additional details about the the actual statistics. The roi is calculated for a box centered at *x,y* if specified. The size of the box, by default, is the last size of the box cursor. The size of the box can be changed with the inclusion of the *xw,yw* options on the command line. The roi is calculated in the ellipse circumscribed by the rectangular roi if the *-e* flag is present. The *-b* flag turns the mode back to a box. The box/ellipse mode is preserved for the next invocation. The mode can also be set with *set*. Values less than the value *roiminimum* set with the *set* command are excluded from the calculations.

****** LOOK-UP TABLE COMMANDS ******

window *[+ | -] win*
Sets the window setting to *win* if no sign is specified. Otherwise, *win* is added or subtracted to the present window setting.

level *[+ | --] lev* Sets the level setting to *lev* if no plus sign or one minus sign is specified. Otherwise, *lev* is added or subtracted to the present level setting.

wl *win lev* Sets the window and level to *win* and *lev*, respectively.

wl -a Sets the window and level based on the minimum and maximum pixel values in the subcanvas. The algorithm is not always perfect, but it gets you to a good starting point for manual adjustment.

wl -r *[low high]*

wl -b *[low high]*

wl -g *[low high]* Specifies a range of pixels to be highlighted in color. The flags 'r', 'b' and 'g' correspond to red, blue and green, respectively. The values *low* and *high* are the inclusive boundaries of the range in units of CT numbers. If the range is not present, then the off/on state of that color will be toggled. If toggled to on, then the previously used range will be used. The default ranges for red, blue and green are respectively [8000,8100], [8200,8300] and [8400,8500]. When ranges, including the gray scale, overlap, the priority of application is as follows from lowest to highest priority: gray scale, red, blue and then green.

wl -o Toggles the outline mode. Fifteen pixel values starting at a digital value of 9025 are set to different color values. The applicatable pixel values are set without consideration of the zero offset. The initial pixel value can be changed using the *outline* variable with the *set* command. The RGB color values for the different pixels values relative to the first value are as follows:

0	255,0,0
1	0,255,0
2	0,0,255
3	255,255,0
4	255,0,255
5	0,255,255
6	255,128,128
7	128,255,128
8	128,128,255
9	0,64,128
10	128,0,64
11	64,128,0
12	64,0,0
13	0,64,0
14	0,0,64

wl -f Sets pixel values greater than the level plus the window over two to black. This mode is called the *fall* mode.

wl -n The highlighting (in red, blue, and green), outline, and fall modes are turned off. This is known as the *normal* mode.

smpte Loads a pattern, which resembles the SMPTE test pattern, into the display. Use the *gamma* command to sent the gamma value to correct the nonlinearities in the display.

gamma *[value]* Sets up a gamma table for the display with the specified value. The use of a gamma table will not affect the output sent via the halftone command. The command will print the current value of gamma when no arguments are specified. Use the *smpte* command to display a pattern that is used as described in the section below entitled "Gamma Correction".

zero *[value]* Sets the offset value subtracted from pixels before reporting values using the cursor and box commands. The command will print the current value of the offset when no arguments are specified. The use of this command is deprecated in favor of setting the option *zero* with *set.*

reverse *[on | off]* With no arguments the command changes reverses "the video". The options "on" and "off" turn the reverse video mode on and off, respectively.

****** HARDCOPY COMMANDS ******

halftone *[options]* By default, the contents of the canvas are sent to a postscript (PS) printer. The name of the printer is defined in the environment variable **PRINTER.** If **PRINTER** is not found, the printer named **lp** is used. The command is also capable of sending PS or encapsulated postscript (EPS) to a file. The following parameters can be used to change the performance of the command:

width=*x*
Sets the width of the output to *x* inches. The default is six inches.

height=*x*
Sets the height of the output to *x* inches. The default is six inches.

xo=*x*

yo=*y* Sets the position (origin) of the lower left corner of the output to be (x,y) (in inches) from the lower left corner of the paper. The default position is (1,1).

-p Prints the output in portrait mode. This is the default orientation.

-l Prints the output in landscape mode.

-a Sets the width (height) based on the aspect ratio of the display when only the height (width) is specified.

-s The status of the command is printed. Note that the values of **width**, **height**, **xo**, **yo**, **gamma**, **of**, **printer**, orientation and output location (file or pipe to printer spooler) are saved between calls to the command.

-f PS or EPS is sent to a file. The default filename is *img.eps.*

-e The output is sent in EPS format to a file. The **xo, yo,** and orientation options are ignored in the EPS mode.

of=*file* The name of the file to which EPS or PS is saved is set to *file.* The **-f** option is set. The suffix *.eps* or *.ps* is added if necessary depending on whether EPS or PS format is selected, respectively. This option can also be selected without the **of=** prefix.

-n The command options are returned to normal. That is, PS is sent to the printer.

printer=*p*
Sets the name of the printer to *p.* The default printer is **lp.** The **-f** flag is also turned off.

printer *[name]* Without any option, the command prints the name of the printer that the **halftone** command uses. With an argument, the name of the printer used by **halftone** is set to *name.*

write *[!]file[.mi] [-s|8|t]*
Write the currently displayed canvas into the *mi*-file named *file* . The suffix *.mi* is added to the name if necessary. The option will not overwrite existing files unless an exclamation point (!) is used as a prefix to the file name. The **-s** flag causes the annotation to be sent along with the image. In this mode, only an eight bit version of the image is saved. Use a window/level setting of 256/0 (or 256/1024 if the **-s** flag is used) to see the same image when it is displayed. The **-8** flag writes the eight-bit buffer associated with the image to the specified file. The **-t** flag writes the eight-bit buffer associated with the image to the specified TIFF-file. The suffix *.tif* is added to the name if necessary.

annotate *[-p x y] text*

annotate *-c*

annotate *-w [!] file*
The screen is annotated with *text.* To position the text, first execute the **annotate** command. Then move cursor into the canvas to the point that you want the lower-left point of the text to be displayed. Finally, depress the left mouse button. A newline can be entered with a \n. Leading and trailing spaces can be entered with a _. If the **-p** flag is present, the next two arguments are the location of where the text will be placed. The **-c** flag indicates that all the annotation should be cleared. **-w** flag creates a file named *file* that will contain an ASCII representation of the present state of the annotation. The file can be sourced with the **source** command to recreate the annotation at a later time.

lc *[options]* By default, the contents of the center row are plotted in an *xplot* window. The command is also capable of sending other lines or columns to *xplot* or to a file. The following parameters can be used to change the performance of the command:

-c Pick up data in a column instead of a row.

-l Pick up data in a row. This is the default mode.

-d Pick up row *or* column from the last deposited position of the cursor. Note that a cursor position has to specified by depressing the left mouse button when the cursor is positioned over the canvas.

-D Pick up row *and* column from the last deposited position of the cursor. Note that a cursor position has to specified by depressing the left mouse button when the cursor is positioned over the canvas.

-? Print out a help message.

of=*file[.da]* The line or column is stored in the file named *file[.da].* The *.da* suffix is added if required. If file is set to *xplot,* the output is plotted directly in an *xplot* window. The program *xplot* has to be running before the call to *lc.*

x=*x* The starting point of a row is set to *x* if in row mode. In this mode, the default starting position is the first point (actually point zero) of the row. In column mode, the column is set to *x* .

y=*y* The starting point of a column is set to *y* if in column mode. In this mode, the default starting position is the first point (actually point zero) of the column. In row mode, the row is set to *y* .

n=*n* The number of points in a line or column is set to *n.* By default all the points from a line or column are read from the starting point.

****** IMAGE MANIPULATION COMMANDS ******

clear *[-a]* Erases the canvas or subcanvas. The *-a* option causes the *reset* and *annotate -c* commands to be invoked so that the complete display is cleared and the annotation is also cleared; the option stands for "all".

flip *[x|y]* Flips the image in the current subcanvas about either the x- or y-axis. The x-axis is the default axis of flipping.

magnify *[-c] [-x|-y]*
By default, the centered image of size one half of the horizontal and vertical extents of the subcanvas is magnified by a factor of two. The resulting image is inserted into the subcanvas. With the *-c* flag, the image is magnified about the last deposited position of the cursor. The cursor location must be in the subcanvas. Bicubic interpolation is used for the magnification. The command can be repeated multiple times for higher magnification. With the *-x* flag, magnification is performed only in the horizontal (x) direction. With the *-y* flag, magnification is performed only in the vertical (y) direction. The magnification can be performed automatically if the *automagnify* option is set via *set.* There is no way to specify horizontal only or vertical only automatic magnification. Pixel replication is used instead of bi-cubic interpolation if the *replicate* variable is set with *set.* The *-c* flag can be used in conjunction with the *-x* and *-y* flags.

transpose Transposes the image in the current subcanvas. The subcanvas must be square.

offset *o* The pixels in the image in the current subcanvas are offset by *o.*

scale *s [c]* The pixels in the image in the current subcanvas are scaled by *s.* If *c* is specified, then the scaling will take place about this value. In other words, the value of *c* is subtracted before the scaling and then added back in after the scaling is performed. The value of *s* can be negative.

invert The pixel values in the present subcanvas are inverted. The inversion takes place about the current value of the level.

measure *[options]*

Distances and angles are measured by pushing the "m" button on the panel. If two/three cursors are deposited, then distance/angle is measured. Cursors are deposited at the end points and previous graphics is erased. The command *measure* is used to to change the performance of the command:

dx=*x* Sets the x (horizontal) pixel size to *x*. The default value is 1.00. This variable can also be set with the *set* command.

dydx=*x*
Sets the ratio between the y (vertical) and x (horizontal) pixels to *x*. The default value is 1.00. This variable can also be set with the *set* command.

-l Draw a line between the two or three deposited cursors.

-c Don't draw cursors at the locations of the two or three deposited cursors.

-e Do not erase all annotations and lines before drawing new lines or cursors.

-r Reset the variables used in this command to their default values.

****** MISCELLANEOUS COMMANDS ******

status *[-a]* Prints the following information in the tty window: the current working directory; the type of image files; the name of the image file the number of images in the file, and the current image displayed, if the a file is open; the current window and level; and the upper left origin and the horizontal and vertical widths of the subcanvas. If the -**a** flag is present, then additional information, which is relevant primarily for debugging, is also printed. Suffixes are added to the window and level when the fall and range modes are used. The suffix */F* is used with the fall mode. The suffix */R=(low,high)* is used with the range mode.

headers *[first last]*
Prints the images headers, where the header consists of the image number, the x- and y-sizes of the image, and the image label. If *first* and *last* are specified, then the headers are printed for this range of images.

help *[command]* With no arguments, a list of the available commands will be printed. Otherwise, the syntax of *command* will be printed. *command* can be minimally abbreviated.

pwd Prints the name of the current working directory in the tty window.

refresh Re-builds the display in case the display gets corrupted. This means that the image is resent to the display buffer and the look-up-table used for window/level is rebuilt.

source *file* The contents of *file* are sourced (read) and treated as commands. The **source** command itself can be nested. The name of the sourced file will be printed along with the normal prompt, **xpic> .**

pause *[message]* The keyboard is locked until the user enters a carriage return. The command is useful when it is included in files that are sourced with the **source** command. The rest of the command line can be used as a message.

#*[comment]* A line beginning with a sharp, "#", indicates that the line is a comment. The sharp is most useful in files that are sourced with the *source* command.

log *[!]file* The commands entered to **xpic** will be saved in a file named *file*. Logging will continue until the end of the session. Commands entered from a sourced file (see the **source** command) are prepended with a "#[name]:" where *name* is the name of the sourced file. The option will not overwrite existing files unless an exclamation point (!) is used as a prefix to the file name.

!*[command]* If the line contains just a single exclamation point, then **csh**(1) will be invoked in the tty window. When the shell is exited, then normal operation of **xpic** will continue. If a command follows the exclamation point, then the command will be passed to **sh**(1) via the **system**(3) command and executed. The output of the command will be displayed in the command window. It is even possible to invoke an editor with a command like "!vi file".

ls*[args]* The command line is passed to **sh**(1) via the **system**(3) command and executed. This command is a shorthand for *!ls [args].*

expression *exp* The mathematical expression given in the string *exp* is evaluated and the result is printed in the tty window. The operations of addition (+), subtraction or unary minus (-), multiplication (*) and division (/) are permissible. The priorities of the operators follows normal programming convention (for example like Fortran or C). The expression can contain parentheses to change the order of evaluation.

exit

<ctrl>-D Terminates program execution. With **exit,** confirmation of termination is requested.

quit Synonym of **exit. port** *[on|off]* With no arguments the command prints the status of the connection. "closed" means that **xpic** is not listening for connections. "waiting" means that **xpic** is listening but that an external program has not yet hooked up. "active" means that communication is taking place with an external program. The option "on" will turn the port on so that **xpic** will start to listen. "off" turns off the port. You cannot turn off a connection if the port is "active".

set *option value*

set *option*

set Sets the values of options (parameters) that are used by other commands. If the command is used without any command line arguments, then all the options and their values will be displayed. There are two types of options: boolean and valued. The valued options are either integer or floating point. Boolean options are either on or off. The state of the option is specified and displayed with the prefix "no". For example, the option *autoreset* enables automatic reset mode and *noautoreset* disables the option. Valued options take values such as integers and strings from the command line. Some of the options can be abbreviated. For example, *autoreset* and *noautoreset* are abbreviated with *ar* and *noar,* respectively. Some options are read-only, meaning that they cannot be set; these options are prefixed with an asterisk (*) when they are displayed. The following options are available, where their abbreviations and negative forms (in the case of boolean options) are also shown:

autoreset,ar,noautoreset,noar
Enables automatic calling of *reset* when a file is opened with *open.* The default is *autoreset.*

autoclear,ac,noautoclear,noac
Enables automatic clearing of the subcanvas when an image is loaded. The default is *noautoclear.* The use of this option can significantly degrade the performance of *cine.*

autowl,awl,noautowl,noawl
Enables automatic setting of the window/level after an image is loaded. The default is *noautowl.*

automagnify,am,noautomagnify,noam
Enables automatic two-times magnification of an image when it is loaded. Thee the *magnify* command for additional details. The default is *noautomagnify.* It is useful to use the *autoclear* function with the *automagnify* function if the source images are smaller than the subcanvas.

wltrack,wl,nowltrack,nowl
Enables tracking of the window/level sliders. Tracking means that the window and level will be updated as the sliders are moved. Without tracking, the window and level are updated only after the slider controls are released. The default mode of operation is enabled if the X11 client and server are the same computer. If not, too much data has to be transmitted across the network and the tracking has too much lag.

replicate,rep,noreplicate,norep
Tells the *magnify* command to use pixel replication instead of bi-cubic interpolation.

boxred,br,noboxred,nobr
Indicates that box cursors should be drawn in red. Otherwise, the lines of the box are XORed into the image.

roiellipse,re,noroiellipse,nore
Sets type of roi - box or ellipse - with *roi* command. The default is *noroiellipse.*

roiminimum,rmin
Pixels below this value are excluded during the calculation of ROI statistics using the *roi* command. This is a binary value and is not subjected to offset by the *zero* value. It is therefore recommended that the offset be set to zero when using this option. The default value is zero.

zero,zero
Sets the value subtracted from all pixel values to *value.* The default value is 1024. The command *zero* can also be used to affect this value.

scrollx,scx
Sets the *x* (horizontal) value of the *scroll* command.

scrolly,scy
Sets the *y* (vertical) value of the *scroll* command.

dx,dx Sets the x (horizontal) pixel size for the *measure* command.

dydx,dydx
Sets the ratio between the y (vertical) and x (horizontal) pixel sizes for the *measure* command.

skip,sk Sets the increment between images in the + *(next)* and - *(previous)* commands. The value also works for the corresponding buttons (+ and -) on the panel. The default of the argument is one. A value of zero disables the commands. Negative values can be used.

outline,ol
Sets the initial digital value of the pixels to be used in the ouline mode. If this value is changed when the outline mode is active, then the *refresh* command has to executed to invoke a new value of the offset. See the *wl* command for additional information.

gamma,gamma,read-only
Displays the valaue of the gamma-correction last set in the *gamma* command.

panel Prints information about the buttons and sliders in the panel.

mouse Prints information about the use of the mouse.

CONTROL PANEL

A control panel is available to execute commands and for displaying status information. The following functionality is provided:

W (Slider)	(Window) Sets the window width. The slider is tracked if the *wltrack* option is set with the *set* command. The window can also be set with the *wl* and *window commands.*
L (Slider)	(Level) Sets the level of the window/level. The slider is tracked if the *wltrack* option is set with the *set* command. The level can also be set with the *wl* and *level* commands.
-	(Minus) Invokes the - command to display the previous image in a file.
+	(Plus) Invokes the + command to display the next image in a file.
u	(Up) Invokes the *scroll* command with the value of *(0,y),* where *y* is the last vertical scroll value. The horizontal scroll value is preserved.
d	(Down) Invokes the *scroll* command with the value of *(0,-y),* where *y* is the last vertical scroll value. The horizontal scroll value is preserved.
l	(Left) Invokes the *scroll* command with the value of *(x,0),* where *x* is the last horizontal scroll value. The vertical scroll value is preserved.
r	(Right) Invokes the *scroll* command with the value of *(-x,0),* where *x* is the last horizontal scroll value. The vertical scroll value is preserved.
m	(Measure) Invokes the *measure* command.
F/C	(Fine/Coarse) Sets the range of the window/level sliders to be a sub-set or complete range of values.
Browse	Launches the GUI browser in the current working directory.
e-	(Environment Decrement) Changes the current environment to the preceding environment. If the current environment is the lowest possible environment, the current environment is set to the highest possible environment. See the **environment** command for further details regarding environments.
e+	(Environment Increment) Changes the current environment to the next environment. If the current environment is the highest possible environment, the current environment is set to the lowest possible environment. See the **environment** command for further details regarding environments.
Env Display	Displays the current environment in the format *e#,* where # is the number of the current environment.
Quit	Quits **xpic,** with confirmation.
Text Message	Various commands display information in this field. When the cursor is moved, the location of the pixel and the value of the pixel are displayed. When a box cursor is used, and the left mouse button is pressed, the mean and standard deviation in the ROI are displayed.

GUI BROWSER FEATURES

The following are descriptions of the components of the GUI browser, their function, and their use.

Go To Button

This button maintains a history list of previously visited directories, without duplication. Right click to access the list. Left clicking selects the button's default location, the user's home directory.

Go To Path

Enter a path on this line and press enter to redirect the browser to another directory.

Current Folder

Displays the folder path that the file browser is listing.

File List

This pane is the central object of the file browser. It contains a list of files and directories contained in the current folder. The files and directories can be accessed by highlighting them and

pressing the *open* button. If the browser is not in *multiselect* mode, double-clicking the file or directory is equivalent to pressing the *open* button.

File Type

This selector allows the user to apply one of three filters to the files listed in the *File List.* The user can select **.mi* to display only files with an "mi" extension, **.lm* to display only files with an "lm" extension, or *All Files* to display all files.

Stay This option controls the behavior of the browser when the *open* action is taken. If this option is selected the browser window will remain open when a file is opened from the browser. If it is deselected the browser will automatically close when a file is opened from within the browser. This option is selected by default.

Environment

This menu is accessible by right clicking. Users are given a choice of *Env 1* through *Env 10.* Selecting one of these will cause files opened by the browser to be opened into the corresponding *environment.* Left clicking this menu selects the default option, *Env 1.*

File Filter

This field enables the user to enter a custom file filter. The filter entered here governs the contents displayed in the file browser. The filter consists of a string containing regular alphanumeric characters and punctuation, and also shell style wildcard characters. These wildcard characters are "* *? [ ] ^*." The * character is expanded to indicate any string of non-wildcared characters, including a null string. The *?* character is more restrictive. It indicates any single non-wildcard character. The *[* and *]* characters can be used in conjunction with the ^ character to indicate a range of characters to include or exclude in matching. Putting a sequence of characters in square braces will cause them to be considered in matching. Surrounding the same string, preceded by the caret, with braces will cause those characters to be excluded from matching. Presing enter or tab applies the filter. Entering no text applies the filter *.

Examples With the following list of files, the results of applying some filter strings are shown.

file1.mi
file2.mi
file3.mi
IMAGE1.mi
IMAGEB.mi
test.lm

Filter: **.lm*
Results:
test.lm

Filter: **[123]???*
Results:
file1.mi
file2.mi
file3.mi
IMAGE1.mi

Filter: *[^f]**
Results:
IMAGE1.mi
IMAGEB.mi
test.lm

Auto Open
When this option is selected, upon opening a file the first image stored in in it is automatically displayed in **xpic** 's display window. This option is selected by default.

Show Hidden
Selecting this option causes hidden files (files starting with a period) to be displayed. By default they are not displayed. If this option is not selected these files will not be considered for matching to filters.

Multiselect
If this option is selected the browser will allow the user to select multiple files for opening. In this mode of operation the double-click feature is disabled. If no files are selected in this mode of operation the open button will be disabled. This option is disabled by default.

The first file opened under this option will be opened into the environment specified on the **Environment** menu. Subsequent files will be opened into subsequent environments. When the maximum environment is reached no further files are opened. A warning is displayed indicating that not all selected files were opened.

COMMAND LINE OPTIONS

The following options are available on the command line to modify the performance of **xpic**. The options can be minimally abbreviated.

[~[user]][/dir1/.../dirn]file[.mi] ...
The file(s) are opened and the first image in the first file is displayed. Shell style wildcard expressions and file lists may be used. This command line option behaves similarly to the *open* command in xpic's tty window, with the following differences: *mi* type files are the default. Therefore *lm* files cannot be opened in this way. Also, the *=arg* option confuses the command parser and therefore cannot be used. It is necessary to escape wildcard characters with a backslash or single quotes for xpic to process them. For a full description of the rest of the functionality of this command see the *open* command documentation, above.

xw=*width* Set the width of the canvas to *width.* The default value is 512. The minimum size is 256.

yw=*height* Set the height of the canvas to *height.* The default value is 512. The minimum size is 256.

size=*s* Sets the size of the canvas to *s* by *s* . This option is the same as using **xw=s** and **yw=s**.

xp=*x* Display the frame containing the canvas, panel and the tty with its upper left corner at horizontal position *x.* The default value is 0.

yp=*y* Display the frame containing the canvas,panel and the tty with its upper left corner at vertical position *y.* The default value is 0.

rows=*r* The number of row in the tty window will be set to *r.* The default value is 8.

window=*w* The initial window setting will be set to *w.* The default value is 256.

level=*l* The initial level setting will be set to *l.* The default value is 0.

-**o** The panel with the sliders for the window and level is not displayed.

-**F** do not fork off (detach) a copy after invocation. This option is used in conjunction with the debugger, **dbx**(1) or **gdb**(1).

-**c** By default, the cursor reporting is enabled. This flag changes the default mode to no reporting. Use the **cursor** command to enable it.

-**z** Normally the program runs in a CT mode in which 1024 is subtracted from all pixel values. When this flag is set, 1024 will not be subtracted. This is equivalent to using *zero=0* on the command line or running the internal command *zero 0.*

zero=*z* Sets the value subtracted from all pixel values to *z*. The *zero* and *set* commands can also be used to set the value when the program is running.

bits=*b* By default, only 14 bits in each pixel are used. This option can be used to change the number of bits to *b*. Pixel values large than 2^bits-1 will be set to this threshold.

-p Enables communication from an external program through a port. Port communication can also be enabled with the **port** command (see above). See below for details on external program communication.

-a enable the **xpic_attach( )** call (see below) and allocate the shared memory required for the call.

port=*s* Specifies the socket to be used for communication with external programs. The default number is listed below in the INTERPROCESS COMMUNICATION section.

-C Enables pseudocolor mode. The look up table is filled with 64 colors. With the default value of zero=1024, you get all 64 colors using window=64, level=-992, and gamma=1. The colors are somewhat randomized so that adjacent values are shown in distinct colors.

printer=*name* Send the output to the printer named *name* instead of the default printer named **lp**.

title=*t* The title in xpic's frame is set to *t* instead of the default "xpic - revised: date".

-s Separates the tty window and the optional control panel from the canvas.

-l Change the file type to LM high energy.

-Z Change the file type to GZMI, gzipped *MI* file.

gamma=*value[/host]*
Sets the default gamma value. If */host* is specified, then the gamma value will be set only if you are displaying in the specified host. It is possible to specify more than one of these options in the files **~/.xpic** and **.xpic.** Specifying gamma without a hostname overrides a setting with a hostname if the former appears after the latter.

Xview's command line resource arguments (see **xview**(1)) are also available. A useful option is ' -**display** which can be used to change the name of the X server.

-S No not use the MIT shared memory extension for X11 under any circumstances. This option is need if xpic is run through SSH.

DEFAULTS

The internal two routines allow for three default mechanisms to specify options. The first method is to create a file **.xpic** in your **HOME** directory. This file is also known as **~/.xpic.** Or the file **.xpic** can be created in the directory from which the program is invoked. In them, one can place options and flags that will be used before parsing your command line. Lines beginning with a '#' are considered to be comments. A '#' also marks the beginning of a comment anywhere else on an input line. Parsing of arguments ends when either an end-of-file is reached or when the line beginning with the string "//" is found. Text found in the file after the line beginning with "//" are keyboard commands that will be processed before giving you access to the display.

As an example, consider the case where the user wishes to to set the size of the canvas to 640 by 320. Also the user wants to enable the *automagnify* feature. The format for the file **~/.xpic** would be:

```
# this is a comment
xw=640
yw=320 #this is another comment
//
set automagnify
```

The second method to set default options is to use the environment variable **XPICARGS.** The format is the same as the input command line.

The command line options set in the previous example can be specified using the following procedure:

For /bin/sh:
$XPICARGS='xw=640 yw=320'
$export XPICARGS

For /bin/csh
$setenv XPICARGS 'xw=640 yw=320'

There is no way to set tty commands in **XPICARGS.** The order of parsing is **~/.xpic, .xpic, XPICARGS,** and finally the command line. The flag **-@,** if present on the command line, will disable the use of the default mechanisms.

GAMMA CORRECTION

Display monitors usually have a nonlinear response to input voltages. Correction is required in order to faithfully display an image. The correction of the nonlinearities has become to be known generically as gamma correction which follows from the name of the variable in the usual equation used to model the nonlinearities. A command called **gamma** is provided to set amount of correction. The following procedure should be followed to determine the gamma correction for a specific monitor.

Display the image using the the *smpte* command.

Set the window width to 100 using *width 100* .

Set the level according to modality mode: for the CT mode (without the -z option) use *level 0 ;* for the MRI mode (with the -z option) use *level 1024* .

Set gamma until the 0/5% and 95/100% regions are visible. The value of gamma is set with the *gamma* command, i.e., *gamma 2.3* . The 0/5% region is to the right of 0% square and is marked 0/5%. The 95/100% region is to the left of 100% square and is marked 95/100%. The gamma values are a function of ambient light and contrast/brightness settings. The gamma values are also machine-dependent.

Install gamma values in **HOME/.xpic** as follows

gamma=v1/machine1
gamma=v2/machine2

etcetera, where v*i*, i=1,2 is gamma for machine name machine*i*.

A gamma can be set for laser (postscript) printer using the **gamma** option with the **halftone** command. Apple laserwriters need *gamma=1* at present.

What follows is additional information about gamma correction which is reprinted (without permission) from J. G. Och, G. D. Clarke, W. T. Sobol, C. W. Rosen, and S. K. Mun, "Acceptance testing of magnetic resonance imaging systems: report of AAPM nuclear magnetic resonance task force no. 6," Medical Physics 19(1), pp. 217-229, 1992, which is adapted from J. E. Gray, K. G Lisk, D. H. Haddick et al, "A test pattern for video displays and hard-copy cameras," Radiology 154, pp. 519-527, 1985.

Clean the visual display with an appropriate cleaner and soft cloth. Include the front and back surfaces of any autoreflective screens, and the front surface of the cathode ray tube (CRT). Reduce the room illumination to the normal viewing level. A room illuminance level of 5 to 10 lux is recommended.

Display the Society of Motion Picture and Television Engineers (SMPTE) digital test pattern. Adjust the window width to just encompass the range of numbers comprising the SMPTE test pattern. Adjust the window level to either the lower or middle value of the window (depending on the particular software), so that the entire test pattern is visible.

Turn the brightness and contrast controls completely counterclockwise. Increase the brightness level until the video master pattern is just visible on the display. Increase the contrast level until the image is bright and clear, and, the 95% and 100% patches are clearly separated. Do not increase the contrast to the point where the alphanumerics are blurred, smeared, or streaked on the display.

Examine the image. The 5% patch should be just visible inside of the 0% patch. The area of the 0% patch should be almost black with raster lines just barely visible. The 95% patch should be visible inside the 100% patch. The alphanumerics should be sharp and clear.

Note that some video monitors do not have adequate "black clamp". This means that the darker areas of the

image may increase in brightness as the contrast is increased. In this case, the brightness level will have to be decreased as the contrast is increased.

COMMUNICATING WITH AN EXTERNAL PROGRAM

Introduction

There are a number of circumstances where you might want to talk to **xpic** from an external program. The purpose of this section is to describe a library of functions that allow communication with an external program.

Before you dive into the functions, you will have to understand some of the structure of **xpic**. There are actually two image buffers. The first buffer contains the 16 bit image data. The second buffer is an 8 bit display area. What you see as the user is the 8 bit display. The data in the 16 bit area is mapped through the window/level look-up-table to generate the 8 bit data. A key is that the 8 bit area is only updated upon command. In this library the two commands that update the 8 bit display are called **xpic_load( )** and **xpic_wl( ).** The steps to update the display are first to write to the 16 bit area with the **xpic_write( )** command and then to update the 8 bit display with the **xpic_load( )** or **xpic_wl( )** commands. If shared memory is available on your system and **xpic** is running on the same host as your application, then you can read (write) directly from (to) the 16 bit buffer. The **xpic_attach( )** function gives you this access.

Use

xpic has an option to tell it to allow communication via an external port. The option is called **-p**. Note that more than one copy of **xpic** can be run at the same time. Normally, only one copy can use the **-p** flag. The command **port** can also be used to control access. Without options, the command will report the status of the port and external programs connected to **xpic**. The command also supports the options **on** or **off** to enable or disable port access. The **port** command line option can be used to change the socket through which communications will take place. By using different values, multiple copies of **xpic** can be run, all with active open ports. The **port** command (not the command line option) also displays the socket number used for interprocess communication.

Summary of Routines

The rest of this note contains a section for each of the routines in the communication library. The file **xpic_port.h** should be included to obtain prototypes for the routines. Here is a summary of the routines:

Name	Summary
xpic_annotate	display annotation (text)
xpic_attach	attach the 16 bit shared memory
xpic_clear	clear the screen
xpic_close	close the connection
xpic_command	run command in tty window
xpic_cursor	return the status of the cursor
xpic_debug	enable debugging the communication's link
xpic_detach	detach the 16 bit shared memory
xpic_exit	causes **xpic** to terminate execution
xpic_line	draw line from the "current point"
xpic_load	transfer the 16 bit image to the display
xpic_message	send a message to the tty window
xpic_mouse	wait for specified mouse button to be pushed
xpic_move	set the "current point" for line drawing
xpic_open	open the communication channel
xpic_read	read a line of the display
xpic_status	return status
xpic_test	test the communication channel
xpic_transpose	transpose rows & columns for reading and writing
xpic_wl	set the window and level
xpic_write	write a line of the display

xpic_annotate

Syntax:

void xpic_annotate(x,y,s)
int x,y;
char *s;

Description: The text contained in *s* is written out starting at location *(x,y).* No check is made to make sure that the text will fit on the screen. A maximum of 132 characters can be written per call.

xpic_attach

Syntax:

(unsigned short) *xpic_attach()

Description: The shared memory address of the 16 buffer used by **xpic** is returned. Error messages will be printed if shared memory is not available on your system or if the **xpic** that you are connected to is not running on the local machine. The user can read and write at will into this memory. Note that when writing, the contents are not visible until **xpic_load( )** or **xpic_wl( )** are run. The size of the buffer can be obtained with the **xpic_status( )** function. The buffer is organized by concatenating the rows with the top row being first. It is the user's responsibility to make sure that access to the buffer is confined to legal addresses. The maximum value that can be written to the buffer is given by the value *mi_max* that is returned by **xpic_status( ).** It is also the user's responsibility to make sure that no pixels exceed this value. **xpic** will probably core dump if you write a value larger than the maximum value. This is because the look-up-table only has *mi_max+1* entries and pixel values are not checked for being in the proper range when the 16 bit buffer is converted to 8 bits. Communication normally takes place asynchronously because commands are sent to **xpic** and no status confirmation is returned. The exceptions are commands that explicitly ask for information such as **xpic_status( )** and **xpic_mouse( ).** You have to be aware of the asynchronous nature of the communication when using the shared memory. As an example, if you first clear the buffer with **xpic_clear( )** and then write to it. It is possible that **xpic_clear( )** will not be finished before you start writing. In this case, you might issue a **xpic_status( )** command before writing.

xpic_clear

Syntax:

void xpic_clear( )

Description: The 8 bit and 16 bit image areas are both cleared (set to zero).

xpic_close

Syntax:

void xpic_close( )

Description: The communication channel that was previously opened with **xpic_open( )** is closed. The channel will also be closed automatically when the external program exits.

xpic_command

Syntax:

void xpic_command(cmd)
char *cmd;

Description: The zero-terminated text that is contained in *cmd* is run in the tty window as a normal command. The following commands cannot be run with this routine: **source**, **pause**, **exit**, **log**, **cd**, **ls**, **cine**, and commands beginning with an exclamation point (!). Note that **xpic**'s concept of the current directory is not the same as your program. Therefore, commands that reference specific files in Unix's hierarchy (like **open**(2)) should use absolute path names.

xpic_cursor

Syntax:

void xpic_cursor(type,x,y,xw,yw)

int *type,*x,*y,*xw,*yw;

Description: The state of the cursor is returned by this command. The variable *type* returns a value of zero if the cursor is turned off (via xpic's command "cursor off"), a value of one for a crosshairs (via "cursor on" & "box off"), and a value of two for a box cursor (via "cursor on" & "box on"). The variables *x* and *y* contain the location of the crosshair or the center of the box the last time the left mouse button was depressed. The variables *xw* and *yw* contain the dimensions of the box cursor.

xpic_debug

Syntax:

void xpic_debug(level)
int level

Description: This command sets the level of debug messages that are printed out on xpic's text window. A level of zero means no messages. Other valid levels are one and two. This command is not intended for general use. The level is reset to zero every time **xpic_open** is called.

xpic_detach

Syntax:

void xpic_detach()

Description: The memory previously attached with the **xpic_attach( )** command is released. It is not too clear if this command is necessary. The **xpic_close( )** and **xpic_exit( )** functions automatically call **xpic_detach( ).**

xpic_exit

Syntax:

void xpic_exit( )

Description: Causes **xpic** to terminate execution.

xpic_line

Syntax:

void xpic_line(x,y)
int x,y

Description: Draw a line from the "current point" to *(x,y).* The "current point" is set to *(x,y)* after the line is drawn. **xpic_move** can also be used to set the "current point". The line is boolean or'ed with the information contained in the canvas.

xpic_load

Syntax:

void xpic_load()

Description: The contents of the 16 bit image buffer are transferred to the 8 bit image area.

xpic_message

Syntax:

void xpic_message(s)
char *s

Description: The message contained in *s* is sent to xpic's text window. The maximum length of the message is 132 characters.

xpic_mouse

Syntax:

int xpic_mouse(b)
int b

Description: The calling program is blocked until a specified mouse button is depressed. The button, *b,* can be zero, one or two, corresponding to the SELECT, ADJUST, and MENU mouse buttons, respectively. For a right-handed mouse, these buttons normally correspond to the left, middle and right buttons, respectively. If the button, *b,* is three, then any mouse button can be pushed. The number of the depressed mouse button is returned.

xpic_move

Syntax:

void xpic_move(int x,int y)

Description: Sets the "current point" for line drawing to *(x,y).* The command **xpic_line** draws the actual lines.

xpic_open

Syntax:

int xpic_open(host,err)
char *host
int err

Description: A communication channel is established with the xpic running on the host specified in *host.* If *host* is NULL or has zero length (i.e, xpic_open(NULL,err) or xpic_open("",err)), then the **xpic** running on the present machine will be used. **xpic** has either to be invoked with the **-p** flag or to have received the *port on* command in order to allow the communication channel to be opened. The routine **xpic_open** has to be called before any other library calls can be made. Because the host can be specified, it is possible to send images across the the network thus creating a poor-person's X-window system. The routine will print an error message and exit if the connection cannot be made if *err* is zero. Otherwise, the routine returns zero for a successful open and one for a failed open. If another external program is already talking to xpic, then your program will hang until the other program issues a **xpic_close( ).** The host can be optionally appended with the string *:s,* where *s* is the socket number through which communications will take place. The default socket number is listed in the INTERPROCESS COMMUNICATION section of this manual. The number *s* should match the number given with the **port** command line option. These features allow multiple copies of **xpic** to be run at the same time, all with active open ports. If the host, *host,* is of the form *:s,* then communications will take place with the **xpic** running on the local host using socket number *s.*

xpic_read

Syntax:

int xpic_read(x,y,data,n)
int x,y,n
unsigned short *data

Description: A line of 16 bit image data is transferred into *data.* The line number is given by *y,* where the zero'th line is the top line in the display. A line is either a row or column in the data array. The choice of row or column is set with the xpic_transpose command. Pixels beginning at column position *x* are read where the first column is column zero. A maximum of 1024 pixels can be read. The call returns the actual number of pixels read. If the values of *x* or *y* are out of the canvas a value of zero will be returned. If there are insufficient pixels on the specified line, then only the available pixels will be transferred. The command is relatively slow because of the length of the packets that are sent across the socket that is used for communication. The function **xpic_attach( )** can be used for faster access to the 16 bit buffer.

xpic_status

Syntax:

void xpic_status(x,y,window,level,mi_max)
int *x,*y,*window,*level,*mi_max

Description: Key parameters that describe the status of xpic are returned. *x* and *y* are the number of columns and rows, respectively, in the image canvas. *window* and *level* are the window and level, respectively. *mi_max* is the maximum value that can be contained in a pixel.

xpic_test

Syntax:

int xpic_test()

Description: This command sends a test packet to xpic to verify the communication channel. The routine returns zero for a passed test and one for a failed test.

xpic_wl

Syntax:

void xpic_wl(window,level)
int window,level

Description: The window and level are set to *window* and *level,* respectively. The 8 bit image area is also updated.

xpic_transpose

Syntax:

void xpic_transpose(dir)
int dir

Description: This command determines the direction that a vector of data is written. The value of *dir* equal to zero and one specifies writing to rows and columns, respectively. The value set by this command stays in affect until the next call to it. The command xpic_open resets *dir* to zero.

xpic_write

Syntax:

void xpic_write(x,y,data,n)
int x,y,n
unsigned short *data

Description: A line of 16 bit image data is transferred from *data* to xpic. The line number is given by *y,* where the zero'th line is the top line in the display. Pixels beginning at column position *x* are written where the first column is column zero. A line is either a row or column in the data array. The choice of row or column is set with **xpic_transpose( )**. A maximum of 1024 pixels can be written. If there are insufficient pixels on the specified line, then only the available pixels will be transferred. The command is relatively slow because of the length of the packets that are sent across the socket that is used for communication. Data that are larger than *mi_max* (see **xpic_status( )**) are set to *mi_max* before being stored. The effect of this command will not be seen on the 8 bit display until a **xpic_wl( )** or **xpic_load( )** is issued. The function **xpic_attach( )** can be used for faster access to the 16 bit buffer.

FILES

libxpic.a

actual name of the library **xpic.a.** The library can be accessed with **-lxpic** when using a C compiler or the linker. When compiling on Solaris, the following additional libraries are required *-lgen -lsocket -lnsl.*

~/.xpic

.xpic initialization (startup) files.

ENVIRONMENT

PRINTER

the default name of the printer that the **halftone** command uses. If not found, the printer named **lp** is used.

XPICARGS

contains command line arguments.

DISPLAY
contains the name of the default X server. The default can be overridden with xview's (**xview**(1)) **-display** command line option or resource. **xpic** examines the name of the server to see if the server and the client are running on the same machine. In this case, MIT's shared memory extension to X11 is used (if available) to speed up certain operations (window and level in particular). Information following a colon (:) in the variable is ignored.

HOME contains shell's concept of your home directory. The **cd** command without arguments will change directories to **HOME.**

INTERPROCESS COMMUNICATION

The program uses a number of sockets for interprocess communication. It might be necessary to change the socket numbers if the program conflicts with other programs that utilize interprocess communication.

8125 Default socket that is used by port connections via the **xpic.a** library. The number can be changed with the **port** command line option.

SEE ALSO

cplot(1), **lc**(1), **plot3d**(1), **qplot**(1), **xplot**(1), **xview**(1), **dami**(1), **armi**(1), **crcplot**(3), **parse**(3), **parse_mi**(3)

AUTHOR

Carl R. Crawford is the original author.

Hara Levy produced the name xpic.

Jim Kohli designed the program's icon.

Owen Dake and Paul Granfors provided the core of the *halftone* command; their code was based on code written by Phillip Ward.

Greg Larson wrote the code for the *smpte* command, and implemented the *-x* and *-y* flags for *magnify.*

Matthew Hirsch provided the colormap for the outline mode in the "wl" command, and the GZMI, gzipped MI file, open functionality, wrote the GUI browser, added buttons to the command pane, added environment control enhancements, open command enhancements, and updated the cine command and documentation. His work is described in the following reports:

Hirsch, M. "Running XPIC and XPLOT under Microsoft Windows using Cygwin," Corporate Imaging Systems Technical Report 03-27, Analogic Corporation, August 29, 2003.

Hirsch, M. W., "XPIC imaging utility graphical user interface file browser and other enhancements," Corporate Imaging Systems Technical Report 03-24, Analogic Corporation, August 20, 2003.

Ibriham Bechwati provided the core code for cine command frame skipping.

BUGS

xpic does not always run correctly on 8-bit displays, especially when there are other big users of the colormap.

Hosts can only be specified by name, not by internet number.

The control frame, when enabled with the **-s** flag, comes up behind the display panel.

Commands entered in the tty window when running under Linux are echoed twice because of a bug in the Linux port of Xview. The bug is related to the *termios* settings of the file descriptor created by the *ttysw* component of the Xview library. Xview improperly echos input lines in the *ttysw* when it takes responsibility for input canonicalization (often called line dicipline code). This could be bypassed by configuring the *ttysw* as a canonical terminal (invoking it with the *TERMSW* keyword, or setting the *ICANON* bit in the

c_lflag mode element of the *termios* structure obtained from *ttysw 's* tty file descriptor by *tcgetattr*). However, the **xpic** keyboard processor would then be responsible for line dicipline code, which is not currently implemented.

The following command must be initially executed in the tty window when running DEC/UNIX on a DEC/ALPHA: *!reset* . Also on the DEC, the scroll bar does not exist for the tty window.

Typing control-C in the command window on some computer systems may cause xpic to die.

Using the environment increment and decrement buttons while a cine command is in progress effectively pulls the carpet out from under the cine command. The command continues to try to load images, but attempts fail in the new environment. The message *Open A File First* or *Bad Image Number* will be repeated for as many image slices the cine command had yet to display. Although this error is merely aesthetic, it is recommended that users refrain from using the environment increment and decrement buttons during cine commands.

Right-clicking in the tty subwindow will bring up a context menu designed for use in an XView textsw application. This artifact allows the user to open, among other things, a text file browsing pane, and find window. It is possible to crash xpic by executing a find and replace procedure from this find dialog. The text editor allows the user to open text files, but has not been fully documented. The commands on the edit menu are functional and may be useful. The use of commands on the history menu can cause erratic behavior. Disabling scrolling may break the link between the keyboard process and xpic. It is recommended that these "features" not be used, as they are unintentional, undocumented, untested, and may have adverse affects on the stability of xpic.

Specifying X11 window arguments does not work under Cygwin.

NAME

xplot – display plot(5) files in the X environment

SYNOPSIS

xplot [*options*]

DESCRIPTION

xplot opens a window and accepts **plot**(5) commands through an inter-process communication socket. **qplot**(1), **cplot**(1) and **plot3d**(1) automatically connect to this socket. The graphics subroutine **plots**(3) (part of **crcplot**(3)) makes the connection to the program via calls to **plot_openpl**(3) that is part of **plotcrc**(3). **xplot** and the application program can be on different machines. The window can be resized to obtain smaller or larger plots. The window can also be converted into an icon in order to save window space. Functionality is included to copy the graphics to secondary windows and to send graphics to POSTSCRIPT printers. All user interfacing is done via panels and pull-down menus that will be described below.

MAIN PANEL AND MENU

Optionally, via the *-p* command line flag, a panel with a number of buttons and a message area is displayed above the graphics area. The buttons are defined as follows:

Erase The screen is cleared. Calls to **erase**(3) (part of **crcplot**(3)) and **plot_erase**(3) (part of **plotcrc**(3)) also clear the screen.

Hard The contents of the main canvas are copied to the POSTSCRIPT printer named *lp*. A pop-up window for the properties of the hardcopy can be obtained if the right mouse button is clicked over the **hardcopy** button. The name of the printer and the size orientation of the plot can be controlled in this pop-up window. See the section below entitled HARDCOPY OPTIONS for additional details.

Refresh The graphics is redrawn.

Copy Causes the graphics to be copied into a new frame. The copied graphics can be printed at a later time. See the section entitled COPY OPTIONS for additional details.

Quit The program is terminated.

A pull-down menu is also available when the right mouse button is depressed while in the plotting area. All of the features provided by the buttons described above are available in the menu. In addition, there is a menu option called **Properties** that will bring up the pop-up panel for the hardcopy options.

HARDCOPY OPTIONS

The following options are available in the **hardcopy** properties pop-up panel:

Printer The POSTSCRIPT output will be sent to the specified printer. The default name is *lp*. The environment variable PRINTER can be used to override the default printer name.

Orientation The graphics will be printed in either the portrait, the default, or in the landscape orientation.

Horizontal Offset

Vertical Offset The bottom left corner of the graphics will be offset from the bottom left (bottom right) corner of the paper by the specified amount in inches when in the portrait (landscape) orientation. The default offset is one half inch.

Scale The size of graphics is specified as a percentage of the the total amount of space available on the page after the offsets are subtracted. The default value is 60%.

Buttons named **Hard, Erase, Refresh,** and **Copy** also exist in the properties pop-up. They have the same functionality as the buttons described above. The only exception is that the **Hard** button always prints the graphics in the main window, not in a copy window. In addition, a **Dismiss** button is available to dismiss the pop-up.

COPY OPTIONS

A pull-down menu is available in the windows (there can be more than one) containing copied graphics. The following options are in the menu:

Hardcopy copies the graphics to the POSTSCRIPT printer. The options described above for printing graphics from the main plotting window are used.

Copy copies the current contents of the main plotting window into the window from where the mouse is depressed.

Properties brings up the pop-up for entering options for hardcopy output.

Refresh the graphics is redrawn.

Quit dismisses the window containing copied graphics.

OTHER FUNCTIONALITY

The postion of the mouse can be extracted in an external program using the **mouse( )** and **plot_mouse( )** commands that are part of the **crcplot**(3) library.

COMMAND LINE OPTIONS

The following options are available on the command line to modify the performance of the program:

printer=*p* Send the output to the printer named *p* instead of the default printer named *lp*. The printer name is not limited to one character.

scale=*s* When the graphics are sent to the line printer, the graphics will be scaled by *s,* where the units are percent. The default value is 60%.

hor=*h* The horizontal offset of the plot is changed from 0.5 to *h* inches.

ver=*v* The vertical offset of the plot is changed from 0.5 to *v* inches.

-p The panel containing the buttons (**Hard, Erase, Copy, Quit** and **Refresh**) will be displayed. Instead a pop-up menu, which is invoked from the canvas, is available. The pop-up panel for the hardcopy options will contain text messages instead of the panel in this case.

-F The program will not fork off a copy of itself after invocation. This option is used in conjunction with the debugger, **dbx**(1).

port=*s* Specifies the socket to be used for communication with external programs. The default number is 8124.

The command line frame arguments of **xview**(1) (**-Ws, -Wp,** etc.) are supported in their long and flag forms.

DEFAULTS

The parse routines internal to the program allow for two default mechanisms to specify options. The first method is to create a file named **~/.xplot.** In it, one can place options and flags that **xplot** will use before it parses your command line.

As an example, consider the case where the user wishes to to set the size of the frame that contains xplot to 640 by 320. The correct format for the file **~/.xplot** would be:

```
-Ws 640 320
```

The second method to set default options is to use the environment variable **XPLOTARGS.** The format is the same as the input command line.

The options set in the previous example can be specified using the following procedure:

For /bin/sh:

```
$XPLOTARGS='-Ws 640 320'
$export XPLOTARGS
```

For /bin/csh

```
$setenv XPLOTARGS '-Ws 640 320'
```

The file **~/.xplot** will be parsed if it exists. Next, **XPLOTARGS,** will be parsed if it exists. Finally, the command line is parsed.

ENVIRONMENT

PRINTER name of default POSTSCRIPT line printer.

XPLOTARGS additional command line arguments for startup

FILES

/var/tmp/xplot.data
intermediate storage for plots

~/.xplot command line arguments for startup The flag **-@,** if present on the command line, will disable the use of the default mechanisms.

SEE ALSO

qplot(1), **xpic**(1), **plot3d**(1), **crcplot**(3), **mouse**(3), **cplot**(1), **plotcrc**(3), **plots**(3), **plot_mouse**(3), **plotps**(1), **lpr**(1), **parse_parse**(3)

AUTHOR

Carl R. Crawford

Malcolm Slaney wrote **sunplot** on which **xplot** is based.

BUGS

The program will die if there is any problems with the **lpr**(1) that is used to send output to the printer.

NAME

parse.a – a C support package

SYNOPSIS

#include <parse.h>

struct PARSE_OPTION_TABLE parse_table;
struct PARSE_FLAG_TABLE parse_flag_table;

DESCRIPTION

This section describes functions that are used to support development of C language programs. The primary purpose of the functions is to support the parsing (hence the name of the library) of the command line. The parsing functions are described in **parse_parse**(3). The key parsing routine is called **parse_basic**(3), which uses the global structures **parse_table** and **parse_flag_table.**

Another set of functions support prompting the user for information from the keyboard. These functions are described in **parse_accept**(3). A complementary set of functions are available to print information on **stdout**. These functions are described in **parse_print**(3).

A set of functions are available to read and write *da*-files and *mi*-files. These functions are described in **parse_da**(3) and **parse_mi**(3).

The majority of the rest of the functions emulate standard system calls with builtin error detection. These functions are described in **parse_disk**(3) and **parse_malloc**(3).

The include file **stdio.h** is included with **parse.h**.

LIST OF FUNCTIONS

Name	*Appears on Page*	*Description*
parse_acf	**parse_accept**(3)	accept a **float** from **stdin**
parse_acfmm	**parse_accept**(3)	accept a **float** from **stdin** with min/max checking
parse_aci	**parse_accept**(3)	accept an **int** from **stdin**
parse_acimm	**parse_accept**(3)	accept an **int** from **stdin** with min/max checking
parse_acm	**parse_accept**(3)	accept a mode (one of string) from **stdin**
parse_acs	**parse_accept**(3)	accept a string or filename from **stdin**
parse_act	**parse_accept**(3)	accept a toggle (yes/no) from **stdin**
parse_all_basic	**parse_parse**(3)	parse file, environment, and *argc/argv*
parse_args	**parse_parse**(3)	parse environment with global **parse_table**
parse_args_basic	**parse_parse**(3)	parse environment using **parse_basic**(3)
parse_args_general	**parse_parse**(3)	parse environment using **parse_general**(3)
parse_args_proc	**parse_parse**(3)	parse environment using specified function
parse_atof	**parse_atof**(3)	**atof**(3) with mathematical expressions
parse_atof_args	**parse_atof**(3)	set parameters for **parse_atof**(3)
parse_atof_error	**parse_atof**(3)	report **parse_atof**(3) errors
parse_basic	**parse_parse**(3)	parse *argc/argv* structure using global structures
parse_comm	**parse_parse**(3)	install one option
parse_comm_aa	**parse_parse**(3)	install one option defined in *argc/argv*
parse_dami	**parse_buffer**(3)	convert between *da*- and *mi*-file data
parse_er	**parse_print**(3)	print one string and exit
parse_err	**parse_print**(3)	print two strings and exit
parse_extract_suffix	**parse_string**(3)	extract a suffix
parse_fcheck	**parse_check**(3)	check **float** ranges
parse_fclose	**parse_disk**(3)	**fclose**(3) with error detection
parse_fflags	**parse_parse**(3)	install one flag in specified flag structure
parse_fft	**parse_fft**(3)	Fast Fourier Transform
parse_fft2d	**parse_fft**(3)	two-dimensional Fast Fourier Transform
parse_file	**parse_parse**(3)	parse initialization file with global function
parse_file_basic	**parse_parse**(3)	parse initialization file using **parse_basic**(3)
parse_file_general	**parse_parse**(3)	parse initialization file using **parse_general**(3)

parse_file_proc	**parse_parse**(3)	parse initialization file using specified function
parse_flag_help	**parse_parse**(3)	print syntax of specified flag table
parse_flags	**parse_parse**(3)	install one flag in **parse_flag_table**
parse_fopen	**parse_disk**(3)	**fopen**(3) with error detection
parse_fread	**parse_disk**(3)	**fread**(3) with error checking
parse_fread_block	**parse_disk**(3)	**fread**(3) at specified block
parse_free	**parse_malloc**(3)	**free**(3) with error checking
parse_frmi	**parse_mi**(3)	frees space allocated by *parse_omi*
parse_from_can_float	**parse_canonical**(3)	convert **float** scalar from canonical form
parse_from_can_floatv	**parse_canonical**(3)	convert **float** vector from canonical form
parse_from_can_long	**parse_canonical**(3)	convert **long** scalar from canonical form
parse_from_can_longv	**parse_canonical**(3)	convert **long** vector from canonical form
parse_from_can_short	**parse_canonical**(3)	convert **short** scalar from canonical form
parse_from_can_shortv	**parse_canonical**(3)	convert **short** vector from canonical form
parse_from_can_ushort	**parse_canonical**(3)	convert **unsigned short** scalar from canonical form
parse_from_can_ushortv	**parse_canonical**(3)	convert **unsigned short** vector from canonical form
parse_from_ican_float	**parse_canonical**(3)	convert **float** scalar from inverse canonical form
parse_from_ican_floatv	**parse_canonical**(3)	convert **float** vector from inverse canonical form
parse_from_ican_long	**parse_canonical**(3)	convert **long** scalar from inverse canonical form
parse_from_ican_longv	**parse_canonical**(3)	convert **long** vector from inverse canonical form
parse_from_ican_short	**parse_canonical**(3)	convert **short** scalar from inverse canonical form
parse_from_ican_shortv	**parse_canonical**(3)	convert **short** vector from inverse canonical form
parse_fseek	**parse_disk**(3)	**fseek**(3) with error detection
parse_fwrite	**parse_disk**(3)	**fwrite**(3) with error checking
parse_general	**parse_parse**(3)	parse *argc/argv* structure using static structures
parse_help_basic	**parse_parse**(3)	print help from global option/flag structures
parse_help_general	**parse_parse**(3)	print help from static option/flag structures
parse_host	**parse_string**(3)	return name of host computer
parse_itoa	**parse_string**(3)	convert **int** to string
parse_malloc	**parse_malloc**(3)	**malloc**(3) with error checking
parse_malloc_char	**parse_malloc**(3)	**malloc**(3) **char** with error checking
parse_malloc_double	**parse_malloc**(3)	**malloc**(3) **double** with error checking
parse_malloc_float	**parse_malloc**(3)	**malloc**(3) **float** with error checking
parse_malloc_int	**parse_malloc**(3)	**malloc**(3) **int** with error checking
parse_malloc_long	**parse_malloc**(3)	**malloc**(3) **long** with error checking
parse_malloc_short	**parse_malloc**(3)	**malloc**(3) **short** with error checking
parse_omi	**parse_mi**(3)	open (examine) an *mi*-file
parse_omi_fe	**parse_mi**(3)	*parse_omi* with error handling
parse_opexist	**parse_parse**(3)	check existence of option
parse_option_help	**parse_parse**(3)	print syntax of specified option table
parse_pcheck	**parse_check**(3)	check **int** ranges
parse_pdt	**parse_print**(3)	print day and time on **stdout**
parse_pparse	**parse_parse**(3)	install option defined in string in specified option table
parse_pparse_aa	**parse_parse**(3)	install option from *argc/argv* context in specified option table
parse_prf	**parse_print**(3)	print **float** on **stdout** with message
parse_pri	**parse_print**(3)	print **int** on **stdout** with message
parse_prm	**parse_print**(3)	print a mode (one of string) on **stdout**
parse_prs	**parse_print**(3)	print string on **stdout** with message
parse_prt	**parse_print**(3)	print toggle (yes/no) with message on **stdout**
parse_rblock	**parse_disk**(3)	read a block (512 bytes) of data
parse_rcsid_print	**parse_print**(3)	prints revision number from RCS Id
parse_rda	**parse_da**(3)	read multiple records of float from a *da*-file
parse_remove_suffix	**parse_string**(3)	remove a suffix
parse_rline	**parse_disk**(3)	read line of text from **stdin**

parse_rmi	**parse_mi**(3)	read an *mi*-file
parse_root	**parse_string**(3)	handle tilde (~) escapes in filenames
parse_suffix	**parse_string**(3)	add a suffix to a filename
parse_suffix_malloc	**parse_string**(3)	add suffix to a filename in new buffer
parse_time_init	**parse_print**(3)	initializes time before **parse_time_print**
parse_time_print	**parse_print**(3)	prints system, user, and elapsed time
parse_to_can_float	**parse_canonical**(3)	convert **float** scalar from canonical form
parse_to_can_floatv	**parse_canonical**(3)	convert **float** vector from canonical form
parse_to_can_long	**parse_canonical**(3)	convert **long** scalar from canonical form
parse_to_can_longv	**parse_canonical**(3)	convert **long** vector from canonical form
parse_to_can_short	**parse_canonical**(3)	convert **short** scalar from canonical form
parse_to_can_shortv	**parse_canonical**(3)	convert **short** vector from canonical form
parse_to_can_ushort	**parse_canonical**(3)	convert **unsigned short** scalar from canonical form
parse_to_can_ushortv	**parse_canonical**(3)	convert **unsigned short** vector from canonical form
parse_to_ican_float	**parse_canonical**(3)	convert **float** scalar from inverse canonical form
parse_to_ican_floatv	**parse_canonical**(3)	convert **float** vector from inverse canonical form
parse_to_ican_long	**parse_canonical**(3)	convert **long** scalar from inverse canonical form
parse_to_ican_longv	**parse_canonical**(3)	convert **long** vector from inverse canonical form
parse_to_ican_short	**parse_canonical**(3)	convert **short** scalar from inverse canonical form
parse_to_ican_shortv	**parse_canonical**(3)	convert **short** vector from inverse canonical form
parse_tolower	**parse_string**(3)	convert string to lower case
parse_toupper	**parse_string**(3)	convert string to upper case
parse_updt	**parse_print**(3)	print day and time on specified stream
parse_uprf	**parse_print**(3)	print **float** on specified stream with message
parse_upri	**parse_print**(3)	print **int** on specified stream with message
parse_uprm	**parse_print**(3)	print a mode (one of string) on specified stream
parse_uprs	**parse_print**(3)	print string on specified stream with message
parse_uprt	**parse_print**(3)	print toggle with message on specified stream
parse_urline	**parse_disk**(3)	read line of text from a stream
parse_uwtext	**parse_print**(3)	print string on specified stream
parse_vread	**parse_da**(3)	read vector from a *da*-file
parse_vwrite	**parse_da**(3)	write vector as a *da*-file
parse_vwrite_float	**parse_da**(3)	write float vector to *da*-file
parse_wblock	**parse_disk**(3)	write a block (512 bytes) of data
parse_wda	**parse_da**(3)	write multiple records of float to a *da*-file
parse_wmi	**parse_mi**(3)	write an *mi*-file
parse_wmi_l	**parse_mi**(3)	write an *mi*-file with header label
parse_words	**parse_string**(3)	break string up into words
parse_wtext	**parse_print**(3)	print string on **stdout**
parse_zbuf	**parse_buffer**(3)	zero out buffer of **short**
parse_zfbuf	**parse_buffer**(3)	zero out buffer of **float**
parse_zibuf	**parse_buffer**(3)	zero out buffer of **int**
parse_zrbuf	**parse_buffer**(3)	zero out buffer of **float** (real)
parse_zsbuf	**parse_buffer**(3)	zero out buffer of **short**

FILES

libparse.a

actual name of the library **parse.a.** The library can be accessed with **-lparse** when using a C compiler or the linker.

SEE ALSO

parse_accept(3), **parse_atof**(3), **parse_buffer**(3), **parse_canonical**(3), **parse_check**(3), **parse_da**(3), **parse_disk**(3), **parse_fft**(3), **parse_malloc**(3), **parse_mi**(3), **parse_parse**(3), **parse_print**(3), **parse_string**(3), **stdio**(3), **fread**(3), **fwrite**(3), **fclose**(3), **fopen**(3), **fseek**(3), **atof**(3), **malloc**(3) **rcs**(1)

AUTHOR

Carl R. Crawford

NAME

parse_aci, parse_acimm, parse_acf, parse_acfmm, parse_acs, parse_act, parse_acm – accept information from stdin

SYNOPSIS

#include <parse.h>

void parse_aci(message,default,v)
char *message;
int default,*v;

void parse_acimm(message,min,max,default,v)
char *message;
int min,max;
int default,*v;

void parse_acf(message,default,v)
char *message;
double default;
float *v;

void parse_acfmm(message,min,max,default,v)
char *message;
double min,max,default;
float *v;

char *parse_acs(message,default,suffix,v)
char *suffix,*message;
char *default,*v;

void parse_act(message,default,v)
char *message;
int default,*v;

void parse_acm(message,default,modes,v)
char *modes,*message;
int default,*v;

DESCRIPTION

These routines prompt the user for information. The basic outline of all the routines is as follows. A message contained in *message* is printed on **stdout** after being prepended with the string ">> ". The default value contained in *default* is printed enclosed in square brackets, "[]", and then followed by a terminating colon, ":". The user can then type a new value followed by a carriage-return. The new value is returned to the user in *v.* If a carriage-return is entered after the colon, the default value, *default,* is retuned in *v.*

When entering numbers for **float** and **int**, a mathematical expression can be used. See the documentation for **parse_atof**(3) for the grammar for the expression. The expression cannot contain any spaces.

parse_aci( ) accepts an **int**. **parse_acf( )** accepts a float.

parse_acimm( ) and **parse_acfmm( )** accept **int** and **float**, respectively, along with performing range checking. The minimum and maximum values, *min* and *max* are also printed out between less-than and greater-than marks, "<>". If the user supplied value is not in range *[min,max],* the user will be prompted again. The string "---------" is printed out before the ">>" part of the prompt to remind the user that the entered value was out of range.

parse_acs( ) accepts a string. In many cases, the string will be a filename. To save typing, the user might want to delete the optional suffix. The routines **parse_suffix**(3) and **parse_suffix_malloc**(3) can be used to add the suffix explicitly. The routines described in **parse_da**(3) and **parse_mi**(3) add the suffixes *.da* and *.mi* automatically. If the string pointer *suffix* is not NULL, then the string contained in *suffix* will be printed, after being surrounded by square brackets, "[]", after the file name. If the string *v* is NULL, then space will be allocated for the returned string. If *v* is not NULL it has to point to a buffer long long enough to contain

the returned string. The address of the resulting string is returned. The maximum length is given by **PARSE_MAX_LINE.**

parse_act( ) accepts a toggle which is also known as a boolean value. The default value, *default,* can contain only zero or one that corresponds to "no" and "yes", respectively. Only the first letter of the response matters and the response is case-insensitive.

parse_acm( ) accepts a "mode", where a mode is one the characters contained in the string *modes.* The default value, *default,* corresponds to *modes[default].* The of list of modes is not displayed. By convention, the modes are incorporated into the message string either as capital letters or enclosed in parentheses. The routine is case sensitive so both upper and lower case letters can be included in *modes.* Here is an example:

```
parse_acm("mode: normal(n), abnormal(a), experimental(e)",1,"nae",&v);
```

Which would result in the following output:

```
>> mode: normal(n), abnormal(a), experimental(e) [a]:
```

SEE ALSO

parse(3), **parse_atof**(3), **parse_string**(3), **parse_da**(3), **parse_mi**(3), **parse_disk**(3), **parse_print**(3)

AUTHOR

Carl R. Crawford

NAME

parse_atof, parse_atof_args, parse_atof_error − replacement for atof(3) that handles mathematical expressions

SYNOPSIS

#include <parse.h>

double parse_atof(s)
char *s;

parse_atof_error()

void parse_atof_args(stream,stop)
FILE *stream;
int stop;

DESCRIPTION

parse_atof( ) is a replacement for **atof**(3) that handles mathematical expressions. The expression contained in *s* is converted to a **double** value. The expression can contain the following operators: addition (+); subtraction (-); division (/); multiplication (*); and unary minus (-). The priority of the operators is identical to the C programming convention. The priority of the operators can be changed with the use of parentheses. The expression cannot contain white space. By default, when an error occurs during the parsing of the string, an error message is printed on **stdout** and the program is exited with a call to **exit**(3).

parse_atof_args( ) is used to change the stream, to which error messages are sent, to *stream.* The default value of *stream* is **stdout.** The second argument of the function, *stop,* controls whether or not **parse_atof( )** exits when an error is detected. Values of zero and one indicate to and to not exit, respectively.

parse_atof_error( ) reports whether an error occurred during the last call to **parse_atof( )**. A return value of zero indicates no error and one indicates an error.

SEE ALSO

parse(3), **atof**(3)

AUTHOR

Carl R. Crawford

NAME

parse_zbuf, parse_zrbuf, parse_dami − work on buffers (vectors arrays) of data

SYNOPSIS

```
#include <parse.h>

void parse_zbuf(v,n)
short int *v;
int n;

void parse_zsbuf(v,n)
short int *v;
int n;

void parse_zrbuf(v,n)
float *v;
int n;

void parse_zfbuf(v,n)
float *v;
int n;

void parse_zibuf(v,n)
int *v;
int n;

void parse_dami(da,mi,ncol,nrow)
float *da;
short int *mi;
int nrow,ncol;
```

DESCRIPTION

parse_zbuf( ) and **parse_zsbuf( )** set the values in an array of **short**, which are contained in *v* and are of length *n,* to zero.

parse_zfbuf( ) and **parse_zrbuf( )** set the values in an array of **float**, which are contained in *v* and are of length *n,* to zero.

parse_zibuf( ) sets the values in an array of **int**, which are contained in *v* and are of length *n,* to zero.

parse_dami( ) converts the floating point array contained in *da* to a short integer array contained in *mi.* The input and output arrays can point to the same starting location. The size of the arrays are *ncol* by *nrow.* The input data are scaled so that zero is mapped to 1024 and are linearly scaled so that the output values fall in the range (1024-128,1024+128). A warning message will be printed if all the elements of the array are the same. The resulting array can be saved as a *mi*-file using **parse_wmi**(3).

SEE ALSO

parse(3), **parse_mi**(3), **parse_da**(3)

AUTHOR

Carl R. Crawford

NAME

parse_from_can_long, parse_from_can_short, parse_from_can_float, parse_to_can_long, parse_to_can_short, parse_to_can_ushort, parse_to_can_float, parse_from_can_longv, parse_from_can_shortv, parse_from_can_ushortv, parse_from_can_floatv, parse_to_can_longv, parse_to_can_shortv, parse_to_can_ushortv, parse_to_can_floatv, parse_from_ican_long, parse_from_ican_short, parse_from_ican_float, parse_to_ican_long, parse_to_ican_short, parse_to_ican_float, parse_from_ican_longv, parse_from_ican_shortv, parse_from_ican_floatv, parse_to_ican_longv, parse_to_ican_shortv, parse_to_ican_floatv − convert scalars and vectors to and from canonical and inverse canonical forms

SYNOPSIS

#include <parse.h>

long parse_from_can_long(long n);
short parse_from_can_short(short n);
unsigned short parse_from_can_ushort(unsigned short n);
float parse_from_can_float(int n);
long parse_to_can_long(long n);
short parse_to_can_short(short n);
unsigned parse_to_can_ushort(unsigned n);
int parse_to_can_float(float n);
void parse_from_can_longv(long *v,int n);
void parse_from_can_shortv(short *v,int n);
void parse_from_can_ushortv(unsigned short *v,int n);
void parse_from_can_floatv(void *v,int n);
void parse_to_can_longv(long *v,int n);
void parse_to_can_shortv(short *v,int n);
void parse_to_can_ushortv(unsigned short *v,int n);
void parse_to_can_floatv(void *v,int n);
long parse_from_ican_long(long n);
short parse_from_ican_short(short n);
float parse_from_ican_float(int n);
long parse_to_ican_long(long n);
short parse_to_ican_short(short n);
int parse_to_ican_float(float n);
void parse_from_ican_longv(long *v,int n);
void parse_from_ican_shortv(short *v,int n);
void parse_from_ican_floatv(void *v,int n);
void parse_to_ican_longv(long *v,int n);
void parse_to_ican_shortv(short *v,int n);
void parse_to_ican_floatv(void *v,int n);

DESCRIPTION

These functions are used to convert scalars and vectors from and to canonical and inverse canonical forms. Canonical form is defined to be Big-endian and inverse canonical form is defined to be Little-endian The routines are portable and will therefore run on both Big-endian and Little-endian machines. The routines only put bytes in the proper order and thus the actual contents are never examined.

The code assumes that shorts (and unsigned shorts), longs, and floats are two, four, and four bytes in length, respectively. No errors are reported if these conditions are not met.

parse_from_can_long( ), parse_from_can_short( ), parse_from_can_ushort( ), and **parse_from_can_float( )** convert, respectively, longs, shorts, unsigned shorts, and floats from canonical form into the local machine representation.

parse_to_can_long( ) parse_to_can_short( ) parse_to_can_ushort( ) and **parse_to_can_float( )** convert, respectively, longs, shorts, unsigned shorts, and floats to canonical form from the local machine

representation.

parse_from_can_longv( ), parse_from_can_shortv( ), parse_from_can_ushortv( ), and **parse_from_can_floatv( )** convert, respectively, long, short, unsigned short, and float vectors of length *n* from canonical form into the local machine representation.

parse_to_can_longv( ), parse_to_can_shortv( ), parse_to_can_ushortv( ), and **parse_to_can_floatv( )** convert, respectively, long, short, unsigned short and float vectors of length *n* to canonical form from the local machine representation.

parse_from_ican_long( ), parse_from_ican_short( ), and **parse_from_ican_float( )** convert, respectively, longs, shorts, and floats from inverse canonical form into the local machine representation.

parse_to_ican_long( ) parse_to_ican_short( ) and **parse_to_ican_float( )** convert, respectively, longs, shorts, float to inverse canonical form from the local machine representation.

parse_from_ican_longv( ), parse_from_ican_shortv( ), and **parse_from_ican_floatv( )** convert, respectively, long, short, and float vectors of length *n* from inverse canonical form into the local machine representation.

parse_to_ican_longv( ), parse_to_ican_shortv( ), and **parse_to_ican_floatv( )** convert, respectively, long, short, and float vectors of length *n* to inverse canonical form from the local machine representation.

Conversion to/from longs only works when longs are four bytes; otherwise an error message will be printed.

SEE ALSO

parse(3), **parse_da**(3), **parse_mi**(3)

AUTHOR

Carl R. Crawford

NAME

parse_pcheck, parse_fcheck – check range of data

SYNOPSIS

#include <parse.h>

void parse_pcheck(v,min,max,message)
int v,min,max;
char *message;

void parse_fcheck(v,min,max,message)
double v,min,max;
char *message;

DESCRIPTION

The value *v* is checked to see if it is in the inclusive range *[min,max]*. If it is, the function returns. Otherwise, the program exits with a call to **exit (3)** after printing a message on **stderr**.

parse_pcheck( ) checks **int** variables.

parse_fcheck( ) checks **float** variables.

SEE ALSO

parse(3)

AUTHOR

Carl R. Crawford

NAME

parse_vwrite, parse_vwrite_float, parse_vread, parse_wda, parse_rda – read and write da-files

SYNOPSIS

```
#include <parse.h>

void parse_vwrite(data,n,type,file,message)
char *data;
char *message,*file;
int n,type;

void parse_vwrite_float(data,n,file)
float *data;
int n;
char *file;

float *parse_vread(data,n,maxn,type,file)
char *data;
char *file;
int *n,type,maxn;

void parse_wda(data,col,row,file)
float *data;
char *file;
int col,row;

float *parse_rda(data,col,row,file)
float *data;
char *file;
int *col,*row;

int BLOCK_FLOAT(n)
int n;

int BLOCK_SHORT(n)
int n;
```

DESCRIPTION

These functions are used to read and write *da*-files. The format of a *da*-file is described in the Appendix of this **man**(1) page. Basically, a *da*-file contains a set of vectors of **float** values in big-endian format. Each vector is padded so that it occupies a multiple of 512 bytes, which is equivalent to being "block aligned". In the nomenclature of *da*-files, a vector is called a record. A one block header, 512 bytes, is at the beginning of a *da*-file to tell a client the size of the vectors and the number of vectors. See the SEE ALSO section for a list of programs that can be used to manipulate or display *da*-files.

For all the functions, the name of the *da*-file is given by *file.* The suffix *.da* is appended to the file name using a call to **parse_suffix_malloc**(3).

parse_vwrite( ) writes a vector of data as a one record *da*-file. The data are contained in the vector *data* which contains **float** elements. The pointer to the array should be type-cast to **(char *).** The length of the vector is given by *n.* The comment contained in *message* is also put in the header of the *da*-file. The field contained in *type* is ignored.

parse_vwrite_float( ) writes a vector of type **float** to a *da*-file. The length of the vector, which is given in *data,* is *n.*

parse_vread( ) reads a vector of data from a one record *da*-file. The data is returned in *data.* If data is NULL, then the data is read into a vector which is acquired through a call to **parse_malloc**(3). The type of data is **float.** If *data* is not NULL, then *maxn* is the length of the buffer *data.* The address of where the data are stored is returned.

parse_wda( ) writes a multi-record *da*-file of type **float**. The data contained in *data* has *col* elements per record and *row* records. The names *col* and *row* correspond to columns and rows of an image.

parse_rda( ) reads a multi-record *da*-file of type **float**. The data is returned in the spaced pointed to by *data,* where the space has to contain *col* elements per record and *row* records. The names *col* and *row* correspond to columns and rows of an image. If *data* is NULL, then sufficient space will be allocated for the data. The function always returns the address of where the data was stored.

SEE ALSO

parse(3), **parse_buffer**(3), **parse_canonical**(3), **parse_malloc**(3), **parse_suffix_malloc**(3), **parse_disk**(3), **qplot**(1), **cplot**(1), **plot3d**(1), **cda**(1), **dami**(1), **daas**(1), **asda**(1), **fmm**(1),

APPENDIX: DA-FILE FORMAT

A *da*-file contains a number of vectors of **float** values. Each vector is also called a record. Each record is padded so that it occupies a multiple of 512 bytes, which is equivalent to being "block aligned". The term block means 512 bytes. A one block header, 512 bytes, is at the beginning of a *da*-file to tell a client the size of the vectors, the number of vectors, and the type of data contained in the vectors.

Let *n* be the length of each vector (all the vectors have to have the same length) and *k* be the number of vectors. The first block of the *da*-file contains an array of 256 **short**'s:

```
short header[256];
```

where

```
header[0] = n;
header[1] = k;
```

The rest of the header can be used by the user for additional storage.

The rest of the *da*-file contains the vectors themselves. The vectors should be written in canonical form. Recall that the vectors have to be padded so that each vector ends on multiples of 512 bytes. The macros **BLOCK_FLOAT**(*n*) return the number of blocks in a vector of length *n.*

AUTHOR

Carl R. Crawford

NAME

parse_fopen, parse_fclose, parse_popen, parse_pclose, parse_fseek, parse_fread, parse_fwrite, parse_rblock, parse_wblock, parse_fread_block, parse_urline, parse_rline – read and write to disk and other streams

SYNOPSIS

```
#include <parse.h>

FILE *parse_fopen(file,mode)
char *file;
char *mode;

void parse_fclose(stream)
FILE *stream;

FILE *parse_popen(cmd,mode)
char *cmd;
char *mode;

void parse_pclose(stream)
FILE *stream;

void parse_fseek(stream,offset,type)
FILE *stream;
long offset;
char *type;

void parse_fread(ptr,size,n,stream)
FILE *stream;
void *ptr;
int size,n;

void parse_fwrite(ptr,size,n,stream)
FILE *stream;
void *ptr;
int size,n;

void parse_fread_block(ptr,size,n,stream,block)
FILE *stream;
void *ptr;
int size,n,block;

void parse_rblock(stream,start,data,count)
FILE *stream;
int start,count;
void *data;

void parse_wblock(stream,start,data,count)
FILE *stream;
int start,count;
void *data;

int parse_urline(stream,s)
FILE *stream;
char *s;

int parse_rline(s)
char *s;
```

DESCRIPTION

These routines are basically calls to the **stdio**(3) routines with the similar name. The main difference with the original functions is that error checking is performed. If an error occurs (for example, not being able to open a specified file) an error is printed on **stderr** and the program is exited via a call to **exit**(3). In the case of a read or write error, the name of the file attached to stream will also be printed along with the error message. Please see the specified **stdio**(3) function for additional details on the arguments.

Some of the functions utilize the concept of blocks. A block is defined to be 512 bytes. In terms of Sun's C-compiler, a block consists of 512 **char**, 256 **short**, or 128 **float** or **int**. A file can be considered to be a series of blocks (if its total size in bytes is divisible by 512). By definition, the first block of a file is number zero.

parse_fopen( ) is a call to **fopen**(3). If *mode* is the write mode, "w", then the file, *file,* will be deleted if it exists. The argument list for this function is identical to the version in **stdio**(3).

parse_fclose( ) is a call to **fclose**(3). The argument list for this function is identical to the version in **stdio**(3).

parse_fseek( ) is a call to **fseek**(3). The first two arguments, *stream* and *offset* are passed directly to **fseek**(3). The last argument, *type,* can be one of the following strings: "b", "c", or "e" which correspond to the "beginning of the file", "the current position", or "the end of the file", respectively. Note that the **stdio**(3) version uses SEEK_SET, SEEK_CUR, or SEEK_END as the last argument.

parse_fread( ) is a call to **fread**(3). The argument list for this function is identical to the version in **stdio**(3).

parse_fwrite( ) is a call to **fwrite**(3). The argument list for this function is identical to the version in **stdio**(3).

parse_fread_block( ) consists of calls two calls, one to **parse_seek( )** and then a second call to **parse_fread( ).** The seek positions the stream at the block given by *block.*

parse_rblock( ) reads *count* blocks of data, starting from block *start,* from stream *stream* into *data.*

parse_wblock( ) writes *count* blocks of data, starting from block *start,* from *data* to stream *stream.*

parse_urline( ) reads a line of text from the specified stream. **parse_rline( )** reads a line of text from **stdin**. The text is returned in *s* with the trailing NEWLINE character stripped off. The maximum number of characters that can be read is defined by **PARSE_MAX_LINE.** The functions return a value of one if an EOF is reached and zero otherwise.

SEE ALSO

parse(3), **stdio**(3), **fread**(3), **fclose**(3), **fseek**(3), **fread**(3), **fwrite**(3)

AUTHOR

Carl R. Crawford

BUGS

parse_fopen( ) does not report when an existing file cannot be deleted.

NAME

parse_fft, parse_fft2d – in-place Fast Fourier Transforms

SYNOPSIS

#include <parse.h>

void parse_fft(a,b,m,mode);
float *a,*b;
int m,mode;

void parse_fft2d(a,b,mx,my,mode);
float *a,*b;
int mx,my,mode;

DESCRIPTION

These functions take the one- and two-dimensional in-place Fast Fourier Transforms (FFT) or the inverse FFT, IFFT, of a complex vector and arrays. The FFT is performed if *mode* is zero and the IFFT if *mode* is one.

parse_fft( ) performs a one-dimensional FFT of a complex vector. The length of the data is two raised to the power *m.* The real and imaginary parts of the complex input and output vector are contained in *a* and *b,* respectively.

parse_fft2d( ) performs a two-dimensional FFT of a two-dimensional complex array. The real and imaginary parts of the complex input and output arrays are contained in *a* and *b,* respectively. The input array is formed by concatenating the rows of original array. The length of a row is given by two raised to the power *mx* and the number of columns is two raised to the power *my.*

SEE ALSO

parse(3)

L. Rabiner and B. Gold, "Theory and application of digital signal processing," Prentice-Hall, 1975, pp. 366-368.

AUTHOR

Carl R. Crawford

NAME

parse_malloc, parse_free, parse_malloc_float, parse_malloc_char, parse_malloc_int, parse_malloc_long, parse_malloc_double, parse_malloc_short – calls to malloc(3) and free(3) with error detection

SYNOPSIS

#include <parse.h>

void *parse_malloc(size_t n)
int n;

void parse_free(p)
void *p;

float *parse_malloc_float(n)
int n;

char *parse_malloc_char(n)
int n;

int *parse_malloc_int(n)
int n;

long *parse_malloc_long(n)
int n;

double *parse_malloc_double(n)
int n;

short *parse_malloc_short(n)
int n;

DESCRIPTION

These routines are basically calls to **malloc**(3) and **free**(3) routines. The main difference with the original functions is that error checking is performed. If an error occurs (for example, not being able to allocate space) an error is printed on **stdout** and the program is exited via a call to **exit**(3).

parse_malloc( ) returns a **char** pointer to a buffer containing *n* **char**.

parse_malloc_float( ) returns a **float** pointer to a buffer containing *n* **float**.

parse_malloc_char( ) returns a **char** pointer to a buffer containing *n* **char**.

parse_malloc_int( ) returns a **int** pointer to a buffer containing *n* **int**.

parse_malloc_long( ) returns a **long** pointer to a buffer containing *n* **long**.

parse_malloc_double( ) returns a **double** pointer to a buffer containing *n* **double**.

parse_malloc_short( ) returns a **short** pointer to a buffer containing *n* **short**.

parse_free( ) frees the space allocated with any of the above functions.

SEE ALSO

parse(3), **malloc**(3), **free**(3)

AUTHOR

Carl R. Crawford

NAME

parse_rmi, parse_wmi, – read, write and manipulate *mi*-files

SYNOPSIS

#include <parse.h>

void parse_wmi(char *file,unsigned short *image,int ncol,int nrow)

void parse_wmi_l(char *file,unsigned short *image,int ncol,int nrow,char *label)

unsigned short *parse_rmi(char *file,unsigned short *image,int *ncol,int *nrow)

PARSE_MIHEAD *parse_omi(FILE *iu,int writeonly)

PARSE_MIHEAD *parse_omi_fe(FILE *iu,int writeonly,FILE *ou,int err)

void parse_omi_checklimit(int limit)

void parse_frmi(PARSE_MIHEAD *mihead)

DESCRIPTION

These functions read, write and manipulate *mi*-files. An *mi*-file contains a number of images. Each image has *ncol* columns and *nrow* rows of 16-bit pixels. At least on a Sparc, a pixel is contained in a *unsigned short.* The size of each image and an image label are contained in an image header that precedes the pixel data. The *mi*-file itself begins with a file header indicating the number of images that follow in the file. The complete specifications for an *mi*-file are described in the Appendix of this **man**(1) page.

parse_wmi( ) writes the image contained in *image* to an *mi*-file. If the filename, *file,* begins with two underscores (_), the underscores will be stripped off and the image appended to the file. The suffix *.mi* is appended to the filename if necessary. The image number (slot) into which the image was written will be printed on *stdout.* The printing is disabled if *ncol* is less than zero. In the case, the number of columns in the image is set to *-ncol.*

parse_wmi_l( ) has the same functionality as **parse_wmi( ).** The only difference is that the label *label* is placed in the image header of the *mi*-file.

parse_rmi( ) reads the last image in an *mi*-file, *file,* into *image.* The suffix *.mi* is appended to the filename if necessary. If *image* is NULL, then space will be allocated for the image and a pointer to the space will be returned. The size of the image is also returned. A message is printed on *stdout* if more than one image is contained in the specified file. The filename can also include the suffixes *:l* or *:L* to indicate that the last image should be read. The suffixes *:f* or *:F* indicate that the first image should be read. The suffix *:n,* where *n* is an integer, indicates that image *n* should be read, where the first image is number one. If a suffix is used, the warning message is not printed.

parse_omi( ) opens (i.e., examines) the *mi*-file that is accessible via the file descriptor *iu.* The function returns a pointer to the type PARSE_MIHEAD that has the following structure:

```
typedef struct {
        int type;               /* type of header: 256, 1024, 65K */
        int n;                  /* number of images */
        int same;               /* nonzero: images have same size */
        int contig;             /* nonzero: images block contiguous */
        int order;              /* nonzero: images sequentially ordered */
        int max_ncol;           /* max ncol for all images */
        int max_nrow;           /* max nrow for all images */
        int min_ncol;           /* min ncol for all images */
        int min_nrow;           /* min nrow for all images */
        int *ncol;              /* array of ncol's per image */
        int *nrow;              /* array of nrow's per image */
        char **label;           /* array of labels per image */
        unsigned long *bptr;    /* array of block pointers to image plus header */
        unsigned long *nblk;    /* array of size of images (in blocks) w/o header */
        unsigned long first;    /* first block used */
```

```
        unsigned long tblk;         /* total blocks in file */
} PARSE_MIHEAD;
```

The structure element *type* corresponds to one of the following types of *mi*-files PARSE_MI_256, PARSE_MI_1024, or PARSE_MI_65K. The former two types are older versions and their use will be phased out over time. The structure element *n* is the number of images in the file. The structure element *same* is nonzero if all the images have the same number of columns and they have the same number of rows. The structure element *contig* is nonzero if the images and headers appear without any gaps in the file with the exception of after the first block, where a block is 512 bytes. The structure element *order* is nonzero if the images are consecutive in the file. Files of type PARSE_MI_65K must be contiguous and ordered. The structure elements *max_ncol, max_nrow, min_ncol,* and *min_nrow* are the maximum and minimum values of the *ncol* and *nrow* for all the images. The structure element *ncol* is an array of the number of columns per image. The structure element *nrow* is an array of the number of rows per image. The structure element *label* is an array of pointers to the label contained in each image header. The structure element *bptr* is an array of the first block number of each image, where the first block is actually the one-block image header. The structure element *nblk* is an array of the number of blocks in each image, excluding the image header. The structure element *first* is the block number of the first image header. The structure element *tblk* is the total number of blocks in the file. If *writeonly* is nonzero, then only the structure elements *type, n, first,* and *tblk* fields are filled in. If any errors are detected with the file while filling in the above structure, a message will be printed and the program is exited.

parse_omi_fe( ) has basically the same functionality as **parse_omi( ).** The exceptions are twofold. Error messages are written to *ou* instead of *stderr.* If *err* is zero, the function will return instead of existing. A *NULL* pointer will be returned in this case.

parse_omi_fe( ) and **parse_omi( )** can take a long time to execute when there are a large number of files in files of type PARSE_MI_MAGIC_65K. This is because all the image headers are read by default in order to obtain the size of each image and its label. You can tell these functions to validate the assumption that all images are the same size. In this case, the size of the first image is computed and then the length of the file is computed with this size. If the actual size of the file matches, then all the images are assumed to have the same size. If the *limit set in* **parse_omi_checklimit( )** is positive, then mi-files with more than *limit* images are checked for this assumption. If met, then the label fields will be set to NULL. If the value of *limit* is negative, then the assumption will not be checked. The default of the limit is -1.

parse_frmi( ) frees the space allocated by **parse_omi_fe( )** and **parse_omi( )** for the PARSE_MIHEAD structure. The function should be called to prevent memory leaks when finished with an *mi*-file.

SEE ALSO

dami(1), **parse**(3), **parse_canonical**(3), **parse_disk**(3), **parse_malloc**(3), **parse_suffix_malloc**(3), **xpic**(1)

APPENDIX: MI-FILE FORMAT

The suffix *mi* is an acronym that means *multiple images.* As the name implies, an *mi*-file contains multiple images. In fact, up to 65535 images can be stored in a single *mi*-file.

In order to understand the structure of *mi*-files, it is necessary to introduce the concept of a block. A block consists of 512 bytes. In terms of Sun's C-compiler, a block consists of 512 **char**, 256 **short**, or 128 **int**. An *mi*-file can be broken up into a series of contiguous blocks. The first block is numbered zero. All blocks in an *mi*-file are 16-bit (i.e., short), unsigned integers. All data are written in canonical form; see **parse_canonical**(3) for additional information.

The first block contains the following information: a *magic* number indicating that it is an *mi*-file, the number of images in the file and the first block of the first image. These three fields are located in the following locations respectively: PARSE_MI_LOC_MAGIC, PARSE_MI_LOC_N and PARSE_MI_LOC_FIRST. The magic number is given by PARSE_MI_MAGIC_65K.

Each image consists of an image header followed by the image itself. The image header is also one block long and contains 256 **short**. The elements located at PARSE_MI_LOC_NCOL and PARSE_MI_LOC_NROW contain the number of elements per row and the number of rows, respectively, where the first element is numbered zero. A label can be placed at the beginning of the header, at location, PARSE_MI_LOC_LABEL. The maximum length of the label, including the null termination, is

PARSE_MI_LEN_LABEL characters.

The rows of the image are then concatenated and stored in contiguous blocks. The number of elements in a picture is padded so that the picture ends on a block boundary, which is called block alignment. The rows are not block aligned. The pixel values themselves are **unsigned short**.

The images and their headers are written consecutively in the file. The only exception is that the first image begins at the block indicated in the header contained in block zero.

In the past, there were two other variations of the header for the *mi*-file. At present all three types are supported, but the use of the older two is deprecated. The format described above is called PARSE_MI_65K. With the first older type, PARSE_MI_1024, the first four blocks of an *mi*-file contain an array of 1024 **unsigned short**. The value of an array element is the block in the *mi*-file that an image begins. The first zero entry indicates the end of the list of images. The number of images in an *mi*-file is the number of elements in the array before the first zero. If there are less than 256 images in the *mi*-file, the 256'st element of the header is set to one. With the second older type, PARSE_MI_256, the header only consists of 256 elements. In order to provide backwards compatibility to this type of file, we assume that older files have at most 255 images. Therefore, the 256'st element would be zero. Because of the use of short integers in the header, the maximum number of images that can be stored in an mi-file is actually less than 1024. For example, only 255 256-by-256 images can be archived.

Here is a sample program that creates an *mi*-file of type PARSE_MI_65K with two images:

```
/* test program to create mi-file with two images */
#include <parse.h>

#define E1 64           /* # elements/row first image */
#define R1 32           /* # rows first image */
#define E2 128          /* # elements/row second image */
#define R2 128          /* # rows second image */

int main()
{
        int n1,n2;
        unsigned short int head[256];                   /* header */
        unsigned short int p1[E1*R1],p2[E2*R2];   /* pictures */
        FILE *fd;

        /* do some thing here to generate images in p1 and p2 */

                                        /* create header for mi-file */
        n1 = (E1*R1 + 255 ) / 256;      /* # blocks first image w/o header */
        n2 = (E2*R2 + 255 ) / 256;      /* # blocks second image w/o header */
        parse_zbuf(head,256);           /* zero header */
        head[PARSE_MI_LOC_MAGIC] = PARSE_MI_MAGIC_65K;          /* magic? */
        head[PARSE_MI_LOC_N] = 2;       /* number of images */
        head[PARSE_MI_LOC_FIRST] = 1;           /* first block of first image */
        parse_to_can_shortv(head,256);  /* convert to canonical format */
        fd = fopen("pic.mi","w");
        parse_fwrite(head,sizeof(short),256,fd);

                                        /* write first image */
        parse_zbuf(head,256);           /* zero header */
        head[PARSE_MI_LOC_NCOL] = E1;       /* set up header for image */
        head[PARSE_MI_LOC_NROW] = R1;
        parse_to_can_shortv(head,256);
        parse_fwrite(head,sizeof(short),256,fd);
```

```
        parse_to_can_shortv(p1,E1*R1);
        parse_fwrite(p1,1,512*n1,fd);

                                        /* write second image */
        parse_zbuf(head,256);           /* zero header */
        head[PARSE_MI_LOC_NCOL] = E2;   /* set up header for image */
        head[PARSE_MI_LOC_NROW] = R2;
        parse_to_can_shortv(head,256);
        parse_fwrite(head,sizeof(short),256,fd);
        parse_to_can_shortv(p2,E2*R2);
        parse_fwrite(p2,1,512*n2,fd);
        return(0);
}
```

Here is a sample program that reads the third image from an *mi*-file without using *parse_rmi* and instead uses *parse_omi:*

```
/* test program to read 3rd image from mi-file w/o parse_rmi */
#include <parse.h>

int main()
{
        FILE *fd;               /* for mi-file */
        unsigned short *a;      /* image store */
        PARSE_MIHEAD *mihead;   /* parse_omi return */
        int ncol,nrow;          /* columns and rows of image */
        int n;                  /* total pixels */
        int fblk;       /* first block in image */

        fd = parse_fopen("../xpic/kak.mi","r");     /* open mi-file */
        mihead = parse_omi(fd,0);       /* examine file contents */
        ncol = mihead->ncol[2];                     /* get image parameters */
        nrow = mihead->nrow[2];
        fblk = mihead->bptr[2] + 1;                 /* +1: walk past header */
        n = ncol * nrow;
        a = parse_malloc_short(ncol * nrow);        /* allocate space for image */
        parse_fread_block(a,sizeof(short),n,fd,fblk);
        parse_frmi(mihead);                         /* free space used by mihead */
        parse_from_can_shortv(a,n);                 /* convert from canonical */
}
```

AUTHOR

Carl R. Crawford

NAME

parse_basic, parse_file_basic, parse_args_basic, parse_all_basic, parse_help_basic, parse_general, parse_file_general, parse_args_general, parse_help_general, parse_flags, parse_fflags, parse_comm, parse_comm_aa, parse_pparse, parse_pparse_aa, parse_opexist, parse_file, parse_args, parse_flag_help, parse_option_help, parse_args_proc, parse_file_proc, parse_file_default, parse_arg_default − parse the command line, initialization files, and environment

SYNOPSIS

#include <parse.h>

void parse_basic(argc,argv)
int argc;
char **argv;
global **struct PARSE_FLAG_TABLE parse_flag_table[];**
global **struct PARSE_OPTION_TABLE parse_table[];**

void parse_file_basic(file)
char *file;
global **struct PARSE_FLAG_TABLE parse_flag_table[];**
global **struct PARSE_OPTION_TABLE parse_table[];**

void parse_args_basic(arg)
char *arg;
global **struct PARSE_FLAG_TABLE parse_flag_table[];**
global **struct PARSE_OPTION_TABLE parse_table[];**

void parse_all_basic(file,arg,argc,argv)
char *file;
char *arg;
int argc;
char **argv;
global **struct PARSE_FLAG_TABLE parse_flag_table[];**
global **struct PARSE_OPTION_TABLE parse_table[];**

void parse_help_basic()
global **struct PARSE_FLAG_TABLE parse_flag_table[];**
global **struct PARSE_OPTION_TABLE parse_table[];**

parse_general(options,flags,fd,argc,argv)
struct PARSE_OPTION_TABLE *options;
struct PARSE_FLAG_TABLE *flags;
FILE *fd;
int argc;
char **argv;

parse_file_general(options,flags,fd,file)
struct PARSE_OPTION_TABLE *options;
struct PARSE_FLAG_TABLE *flags;
FILE *fd;
char *file;

parse_args_general(options,flags,fd,arg)
struct PARSE_OPTION_TABLE *options;
struct PARSE_FLAG_TABLE *flags;
FILE *fd;
char *arg;

void parse_help_general(options,flags,fd)
struct PARSE_OPTION_TABLE *options;
struct PARSE_FLAG_TABLE *flags;

FILE *fd;

void parse_flags(s)
char *s;
global **struct PARSE_FLAG_TABLE parse_flag_table[];**

int parse_fflags(flags,s,stream)
struct PARSE_FLAG_TABLE *flags;
char *s;
FILE *stream;

int parse_comm(s)
char *s;
global **struct PARSE_OPTION_TABLE parse_table[];**

int parse_comm_aa(argc,argv)
int *argc;
char ***argv;
global **struct PARSE_OPTION_TABLE parse_table[];**

int parse_pparse(options,s,stream)
struct PARSE_OPTION_TABLE *options;
char *s;

int parse_pparse_aa(options,stream,argc,argv)
struct PARSE_OPTION_TABLE *options;
FILE *stream;
int *argc;
char ***argv;

int parse_opexist(op)
char *op;

void parse_file(file)
char *file;
extern void parse();

void parse_args(arg)
char *arg;
extern void parse();

void parse_flag_help(flags,stream)
struct PARSE_FLAG_TABLE *flags;
FILE *stream;

void parse_option_help(options,stream)
struct PARSE_OPTION_TABLE *options;
FILE *stream;

int parse_args_proc(options,flags,fd,arg,proc,proc_type)
struct PARSE_OPTION_TABLE *options;
struct PARSE_FLAG_TABLE *flags;
FILE *fd;
char *arg;
int (*proc)();
int proc_type;

int parse_file_proc(options,flags,fd,file,proc,proc_type,home,efile)
struct PARSE_OPTION_TABLE *options;
struct PARSE_FLAG_TABLE *flags;
FILE *fd;
char *file;

int (*proc)();
int proc_type;
int home;
int efile;

void parse_file_default(file)
char *file;

void parse_arg_default(arg)
char *arg;

DESCRIPTION

The purpose of the functions described in this section is to help the user easily parse options passed on the command line when a program is executed. A set of routines, called a parser, is provided to support the parsing. The parser is designed with the assumption that all command line arguments are optional. In other words, the program will do something useful if the command line in empty. The concept of having a default execution mode is intrinsic to most commands found in the UNIX operating system. When command line arguments are present, they alter the default operational mode.

The parser has evolved over many years of use and therefore many application programs written by the author of the parser use this functionality. In order to provide backwards compatibility with older application code, there are many versions of the parser. The majority of this documentation will deal with the latest (and best!) generation of the parser. The older versions of the parser will be briefly described in order to help maintain older application code.

The parser support two types of objects: *flags* and *options.* A *flag* is a string beginning with a minus sign (-) and followed by a set of characters. The characters following the minus sign are also known as *flags.* At least one flag must be present after the hyphen. The parser maintains in a structure a list of valid flags and actions to take when a flag is encountered. Typically, a variable is set to one that the flag was present on the command line. Usually the string of flags consists only of upper- and lower-case letters. Be aware that some characters might have to be escaped to prevent interpretation by a shell. The question mark (?), when it is part of a flag, signifies that help is requested (more information is provided below on its use). Examples of valid flags are *-a, -Bcd,* or *-?.* The grammar is set up so that flags are processed from left to right on the command line. Therefore, *-ab* is equivalent to *-a -b* .

The effect of some flags can be *negated,* where negation means that flag will be effectively unset. When a minus sign (-) is encountered in the flags, excluding the minus sign that denotes a sets of flags, all flags afterwards will be negated. When a plus sign (+) is encountered, negation is turned off. In the example *-abc+de,* the flags *a, d* and *e* will be set and flags *b* and *c* will be unset.

An *option* has the form *name=value,* where *name* is the of a parameter and *value* is a number or string to be assigned to *name.* The *name* is also referred to as an *option.* The parser maintains in a structure a list of options and actions to take if the option is present on the command line. Typically, a value is assigned to a variable or a string is copied to a buffer. A common use of options is to specify the names of input and output files. Most of the functions found with the parser allow the *name* and *value* fields to be two separate words with the equals sign (=) still attached to the *name.* This functionality is provided so that filename completion can be used with the shell. The parser is set up to process options from left to right. In general it is possible to repeat the option on the command line and have only the last one count. This feature is useful with the history mechanisms in shells. A single question mark (?) indicates that help is requested (more details below). Most of the parser functions support the question mark without a trailing equals sign. Options can be be abbreviated to the fewest number of characters that will not cause ambiguity in the parser. Flags and options can be mixed on the command line.

The arguments on the command line are passed to the **main** program using the *argc/argv* syntax as follows:

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

where *argc* is the number of arguments, including the command name itself, and *argv* is a pointer to a list

of pointers to the arguments. **parse_basic( )** is the easiest way to call the parser. Given the *argc/argc* interface the parser is called as follows:

```
parse_basic(argc,argv);
```

As stated above, two structures are used in the parsing of the flags and options. The names of the structures are **parse_table** and **parse_flag_table**. They have to be *global* variables for the parser to load. The structures are described in the next two sub-sections.

Flag Structure

Before describing the flag structure in detail, consider the following example of a flag structure:

```
static gauss=0;          /* 1=use gaussian test pattern */
static edata=0;          /* 0= simulated 1=experimental data */
static print=1;          /* 1=print program status */
static myhelp();

struct PARSE_FLAG_TABLE parse_flag_table[] = {
        {'g',&gauss,PARSE_FLAG_SET,"gaussian test pattern"},
        {'e',&edata,PARSE_FLAG_SET,"experimental data"},
        {'p',&print,PARSE_FLAG_CLR,"don't print program status"},
        {'?',(int *)(myhelp),0,0},
        {0,0,0,0}
};
```

When the parser is run, *-g* will set the variable *gauss* to one. Likewise, *-e* sets *edata* to one and *-p* sets *print* to zero. The flag *-?* will cause the routine *myhelp* to be called. The line containing *{0,0,0,0}* is required to tell the parser where the list ends.

The flag structure contains an entry per valid flag plus a "zero" entry to indicate its end. The structure definition, which can be found in **parse.h**, is:

```
struct PARSE_FLAG_TABLE{
        char flag;
        int *var;
        int type;
        char *help;
};
```

The entry **flag** is the one character name of the flag as it will appear on the command line. A special case is the setting of **flag** to *'?'*. In this case, the second argument, **var**, contains the address of a function to call when the question mark appears in a flag. No arguments are passed to the function. If the *'?'* is not present, then the routine **parse_flag_help**(3) is called and the program is exited. A minus sign (-) and a plus sign (+) cannot be used as flags because they indicate *negation* and *end negation,* respectively, to the parser. A space (' ') also cannot be used as a flag as it is ignored in order to support **parse_all_basic**(3).

The entry **var** is the address of the **int** that stores the result of the parser. In two cases **var** is also the address of a function. One case is the use the *'?'* flag. The other case is the use of **PARSE_FLAG_PROC** in the **type** field (see below).

The entry **type** tells the parser what to do to the variable **var**. The following types are valid:

PARSE_FLAG_SET	sets the variable **var** to one. If *negation* is in effect, the variable is set to zero.
PARSE_FLAG_CLR	sets the variable **var** to zero. If *negation* is in effect, the variable is set to one.
PARSE_FLAG_INC	increments the variable **var** by one. An error will occur if *negation* is in effect.
PARSE_FLAG_DEC	decrements the variable **var** by one. An error will occur if *negation* is in effect.

PARSE_FLAG_VALUE(*x*) sets the variable **var** to *x*. An error will occur if *negation* is in effect.

PARSE_FLAG_PROC calls the procedure whose address is in **var**. Be sure to typecast the procedure name to **(int *)** in the flag structure. The flag that caused the procedure to be called is passed to the procedure. An error will occur if *negation* is in effect.

The entry **help** is an optional string that is printed out by **parse_flag_help**(3).

Option Structure

Before describing the option structure in detail, consider the following example of an option structure:

```
static int m=128;
static int o=3;
static float s=0.0;
static char *filename="junk";
int myhelp();

struct PARSE_OPTION_TABLE parse_table[] = {
        {"m",PARSE_INT,(PARSE_ARG)&m,"number of vector points" },
        {"o",PARSE_INT,(PARSE_ARG)&o,"polynomial order"},
        {"shift",PARSE_FLOAT,(PARSE_ARG)&s,"shift in samples"},
        {"ifn",PARSE_CHAR,(PARSE_ARG)&filename,"name of input file"},
        {"?",PARSE_PROC,(PARSE_ARG)myhelp},
        {0,0,0,0}
};
```

When the parser is run, *m=256* will set the variable *m* to 256. Likewise, *o=-4* sets *o* to -4 and *shift=3.4* sets *s* to 3.4. The option *ifn=foo* will cause the pointer *filename* to point to the string *foo* . An option than begins with a question mark (?) will cause the routine *myhelp* to be called. The line containing *{0,0,0,0}* is required to tell the parser where the list ends.

The option structure contains an entry per valid option plus a "zero" entry to indicate its end. The structure definition, which can be found in **parse.h**, is:

```
typedef void **PARSE_ARG;
struct PARSE_OPTION_TABLE {
        char *label;
        int type;
        PARSE_ARG pointer;
        char *help;
        };
```

PARSE_STRUCT is a synonym of **PARSE_OPTION TABLE.** It is used in a lot in older application code. Its use is now deprecated.

The entry **label** is the filename of the option as it will appear on the command line. However, the option can be abbreviated on the command line to the fewest number of characters that will not cause ambiguity in the parser. The label does not have to be the same as the name of the variable put in the **pointer** field. However, for ease of use, it helps if they are the same. A special case is the setting of **label** to *?* In this case, the third argument, **pointer**, contains the address of a function to call when the question mark appears in a flag. If the *'?'* is not present, then the routine **parse_option_help**(3) is called and the program is exited. Another special case is the use of = as a **label**. In this case, a command line argument that is missing an equals sign is processed by this entry. Note that an option can terminate in an equals sign and the value being the next word on the command line. The "=" label is not invoked in this case.

The entry **type** tells the parser what to do to the variable **pointer**. The following types are valid:

PARSE_CHAR causes **pointer** to point to the string contained in *value*. The parser allocates the memory for the string itself. **PARSE_CHAR_POINTER** is a synonym.

PARSE_STRING
causes the string contained in *value* to be copied to the address contained in **pointer.** The buffer into which the string is copied has to be large enough to contain *value.* **PARSE_CHAR_BUF** is a synonym.

PARSE_INT sets the **int** pointed to by **pointer** to *value.* A mathematical expression can be used for *value.* See **parse_atof**(3) for the expression grammar.

PARSE_FLOAT sets the **float** pointed to by **pointer** to *value.* A mathematical expression can be used for *value.* See **parse_atof**(3) for the expression grammar.

PARSE_LONG sets the **long** pointed to by **pointer** to *value.* A mathematical expression can be used for *value.* See **parse_atof**(3) for the expression grammar.

PARSE_TOGGLE
sets the **int** pointed to by **pointer** to either zero or one, depending on the contents of *value.* Zero is used if *value* is n, no, N, NO, or 0 (zero). One is used if *value* is y, yes, Y, YES, or 1 (one).

PARSE_PROC calls the procedure whose address is in **pointer**. The non-abbreviated name of the option that caused the procedure to be called and its argument are passed to the procedure. The prototype for the procedure is:

```
void proc(char *option,char *argument);
```

PARSE_SET sets the **int** pointed to by **pointer** to one. This entry is used to set "side effects"; see below for more information.

PARSE_CLR sets the **int** pointed to by **pointer** to zero. This entry is used to set "side effects"; see below for more information.

PARSE_INC increments the **int** pointed to by **pointer** by one. This entry is used to set "side effects"; see below for more information.

PARSE_DCR decrements the **int** pointed to by **pointer** by one. This entry is used to set "side effects"; see below for more information.

PARSE__VALUE(*x*)
set the **int** pointed to by **pointer** to *x.* This entry is used to set "side effects"; see below for more information.

The entry **pointer** is the address of the variable that stores the result of the parser. In two cases **pointer** is also the address of a function. One case is the use the *'?' entry.* The other case is the use of **PARSE_PROC** in the **type** field. The entry must be typecast to **PARSE_ARG** for reasons of portability and to make **lint**(1) happier.

The entry **help** is an optional string that is printed out by **parse_option_help**(3).

Side Effects

Consider the case of a program might have different "modes" of operation with one of the modes being the default. The different modes would typically be invoked by having flags on the command line specify the different modes. Each of the different modes might also use variables that are exclusive to a specific mode. These variables might be settable using options on the command line. It makes sense that if a variable specific to a mode is set with an option, then the mode should be set as if the corresponding flag was also set on the command line.

Another case to consider is using a set of related variables that are settable with options. It would be better to provide a method to set each option individually or all of them at the same time.

In both of these situations we desire "side effects" when setting an option or a flag. There are number of methods to handle these cases. One way, which is now deprecated, is to use the **PARSE_PROC** or **PARSE_FLAG_PROC** modes and set up a routine to do all the setting of variables. Unfortunately, this method generates code that is hard to read and some of the functionality of the option handler has to be

replicated in the call-back procedure.

Another way is to set the variables corresponding to the options to some unused value (say -1). After the parser has run, you check to see if the variables have been set to another value in which case you can program in the side effects. This method does not work well when combined with the history mechanism of a shell. In other words the parser does not parse from left to right.

Yet another way to handle these situations is to use the older versions of the parser that are described below. The older versions are more versatile but much more application code has to be written. A "better" bad way is to make use of the **parse_opexist( )** function, which is described below.

The "best" way to have side effects is to have multiple entries in the flag or option structures for a given flag or option. When a flag or option with multiple entries is processed by **parse_basic( )** all the entries are executed. The following example illustrates this functionality:

```
static a=0;
static b=0;
static int m=128;
static float xlen=10;
static float ylen=10;

struct PARSE_FLAG_TABLE parse_flag_table[] = {
        {'a',&a,PARSE_FLAG_SET,"set a"},
        {'b',&b,PARSE_FLAG_SET,"set a and b"},
        {'b',&a,PARSE_FLAG_SET},
        {0,0,0,0}
};

struct PARSE_OPTION_TABLE parse_table[] = {
        {"m",PARSE_INT,(PARSE_ARG)&m,"number of vector points" },
        {"m",PARSE_SET,(PARSE_ARG)&b},
        {"xlen",PARSE_FLOAT,(PARSE_ARG)&xlen,"length of x-axis"},
        {"ylen",PARSE_FLOAT,(PARSE_ARG)&ylen,"length of y-axis"},
        {"len",PARSE_FLOAT,(PARSE_ARG)&ylen,"length of both axes"},
        {"len",PARSE_FLOAT,(PARSE_ARG)&xlen},
        {0,0,0,0}
};
```

When the flag *-b* is used, both the *a* and *b* variables will be set to one. When the *m=value* option is used, *m* will be set *value* and *b* will be set to one. The options *xlen* and *ylen* set the values of *xlen* and *ylen,* respectively. However, the option *len* sets both *xlen* and *ylen.*

Initialization files, Environment, Help

parse_file_basic( ) can be used to put command line arguments in an initialization file. If *file* begins with a slash (/), then an attempt will be made to open the specified file. Otherwise, the user's root directory is extracted from the **HOME** environment variable and prefixed with another slash to the user-specified file. If the file can be opened, then command line arguments are read from the file and sent to **parse_basic( )**. Arguments are read until the end-of-file or until a line beginning with // is encountered. Lines beginning with a sharp (#) are comments. Any text after a sharp, including the sharp itself, is also a comment and is stripped off. An arbitrary number of arguments can appear on the same line. By convention, the name of the initialization file should be the name of the program with a period (.) prefixed and perhaps with the suffix *rc* appended. When the filename does not begin with a slash and the local (current working) directory is not the **HOME** directory, the local directory is also examined for the specified file and parsed.

parse_args_basic( ) can be used to put command line arguments in an environment variable. If the environment variable *arg* exists, it is parsed into words using **parse_words**(3) and passed to **parse_basic( )**. By convention, the name of the environment variable should be the name of the program in upper-case and perhaps with the suffix *RC* appended. By convention, initialization files should be parsed first, then environment variable, and finally the command line.

parse_all_basic( ) is basically a combination of calls to **parse_file_basic( ), parse_args_basic( ),** and **parse_basic( )** in that order. If *file* is NULL then **parse_file_basic( )** will not be called. If *arg* is NULL then **parse_args_basic( )** will not be called. The flag **-@,** if present on the command line, will disable the calls to **parse_file_basic( )** and **parse_args_basic( ).**

parse_help_basic( ) is used to print the help information contained in the option and flag structures. In the case of the option structure, the current value of the variables are also displayed. A common use of this function is with the question mark (?) fields in the flag and option structures. The names of the file and and the environment variables used for initialization are also printed. In general, the initialization file and variable are obtained from the call to **parse_all_basic( ).** The file and variable can be respectively set via calls to **parse_file_default( )** and **parse_arg_default( ).**

Complete Example

In this section, a complete program is presented. The program's name is *exp.c* and is used to add, subtract or multiply two numbers. The default mode is to add two numbers, *x* and *y,* whose initial values are 10 and 20, respectively. Here is the complete program:

```
#include <parse.h>
static float x=10,y=20;
static mode=0;     /* 0=add 1=subtract 2=multiply */

struct PARSE_FLAG_TABLE parse_flag_table[] = {
        {'a',&mode,PARSE_FLAG_CLR,"add"},
        {'s',&mode,PARSE_FLAG_SET,"subtract"},
        {'m',&mode,PARSE_FLAG_VALUE(2),"multiply"},
        {'?',(int *)(parse_help_basic),0,0},
        {0,0,0,0}
};

struct PARSE_OPTION_TABLE parse_table[] = {
        {"x",PARSE_FLOAT,(PARSE_ARG)&x,"x"},
        {"y",PARSE_FLOAT,(PARSE_ARG)&y,"y"},
        {"?",PARSE_PROC,(PARSE_ARG)parse_help_basic},
        {0,0,0,0}
};

main(argc,argv)
int argc;
char **argv;
{
        parse_all_basic(".exp","EXP",argc,argv);

        if(mode == 0) printf("addition: %f",x+y);
        else if(mode == 1) printf("subtraction: %f",x-y);
        else printf("multiplication: %f",x*y);
}
```

Generalized Parsing

As seen above, the "basic" parsing functions exit anytime an error is detected. The error messages are always written to *stderr.* The option and flag structures are passed using global variables. These attributes, while fine for most application programs, can cause troubles for interactive programs that support many different internal functions or, in a sense, programs within programs. An example of an interactive program that would have trouble with the basic parsing functions is **xpic**(1).

A set of "generalized" parsing functions are included to support parsing in interactive programs. The following functions are available: **parse_general( )**, **parse_file_general( )**, **parse_args_general( )**, and **parse_help_general( )**. The flag and option structures are passed as arguments. The stream to which

errors are written is also passed as an argument. In case of an error, the functions return a value of minus one.

There is no way to use **parse_help_general( )** directly in the option and flag structures. An intermediate procedure has to be used.

Other Parser Functions

Some of the functions described in this section are the building blocks of the "basic" parser functions. You might to call them to get more control of the parser. Also, they are used by some older application programs.

parse_flags( ) installs the flags contained in the string *s* in the globally defined flag structure **parse_flag_table**. The string must begin with a hyphen. If an error occurs, the program exits. Error messages are sent to **stderr**. **parse_fflags( )** performs the same function with three exceptions: the flag structure is supplied as an argument; error messages are sent to **stream**; and a minus one (-1) is returned in case on an error.

parse_comm( ) installs one option, which is found in the string *s,* in the global option structure **parse_table.** If an error occurs, the program exits. Error messages are sent to **stderr**. The number of the entry (the first entry being one) is returned. The function will not return the correct entry number if multiple entries are used to get side effects. **parse_comm_aa( )** performs the same function with exception that the options are contained in the *argc/argv* context. Options can be split into two words as long as the first word ends with an equal sign (=). **parse_pparse( )** performs the same function as **parse_comm( )** with the exception that the options table is supplied as an argument; error messages are sent to the specified stream; and the function returns a value of minus one (-1) in case of an error. **parse_pparse_aa( )** performs the same function with the exception that the options are supplied in the *argc/argv* context.

parse_opexist( ) returns a boolean value if the option given in *op* has been found by the parser. The the existence table can be reset by passing NULL as the argument. The full name, not abbreviated, of the option has to be supplied. To check for an option passed without a label, call the function with "=" as an argument.

parse_file( ) and **parse_args( )** supply the same functionality as **parse_file_basic( )** and **parse_args_basic( )** with the exception that a globally defined function **parse( )** is called instead of **parse_basic( )**. Here is the prototype of the call-back function:

```
void parse(argc,argv)
int argc;
char **argv;
```

Note that the first value in *argv* will be unused.

parse_file_proc( ) and **parse_args_proc( )** supply almost the same functionality as **parse_file_basic( )** and **parse_args_basic( )** with the exception that the call-back function is supplied in argument *proc* is called instead of **parse_basic( )**. The call-back function has one of two different prototypes that are specified using *proc_type.* Note that the first value in *argv* will be unused. Also note that the function will have to by typecast using **(void (*)()) for** *proc_type* equal to zero. Here is the prototype of the call-back function for *proc_type* equal to zero:

```
void parse(argc,argv)
int argc;
char **argv;
```

Here is the prototype of the call-back function for *proc_type* equal to one:

```
int parse(options,flags,fd,argc,argv)
struct PARSE_FLAG_TABLE *options;
struct PARSE_OPTION_TABLE *flags;
FILE *fd;
int argc;
char **argv;
```

The function should return a minus one in case of an error. Error messages are sent to *fd.* The function

parse_file_proc( ) has two extra arguments *home* and *efile*. If *home* is nonzero, then two files will be examined in the *HOME* directory and the local directory. If *efile* is zero, then missing files will be silently ignored, whereas if the argument is one, then errors will be reported.

prints the help information from the specified flag structure on the specified stream.

parse_option_help( ) prints the help information from the specified option structure on the specified stream. Only the help string for the first occurrence of an entry in the options structure is printed out. An option is printed out only if the help string is present. Help for a lone equals sign (=) entry is printed out as **(empty) equivalent to:**

ENVIRONMENT

HOME Used by the functions that parse the initialization file to obtain the path of the user's home directory.

PROGRAM[ARGS|RC]

Name of the environment variable that contains arguments for the program named *program*. The suffixes **ARGS** and **RC** are used sometimes by convention.

FILES

.program[rc]

Name of the initialization file that contains arguments for the program named *program*. The suffix **rc** is used sometimes by convention.

SEE ALSO

parse(3), **parse_atof**(3), **parse_words**(3), **xpic**(1)

AUTHOR

Carl R. Crawford

NAME

parse_er, parse_err, parse_prf, parse_pri, parse_prm, parse_prs, parse_prt, parse_rcsid_print, parse_time_init, parse_time_print, parse_updt, parse_uprf, parse_upri, parse_uprm, parse_uprs, parse_uprt, parse_uwtext, parse_wtext − print information on stdout, stderr or a specified stream

SYNOPSIS

```
#include <parse.h>

void parse_upri(stream,message,v)
FILE *stream;
char *message;
int v;

void parse_pri(message,v)
char *message;
int v;

void parse_uprf(stream,message,v)
FILE *stream;
char *message;
double v;

void parse_prf(message,v)
char *message;
double v;

void parse_uprs(stream,message,v)
FILE *stream;
char *message;
char *v;

void parse_prs(message,v)
char *message;
char *v;

void parse_uprt(stream,message,v)
FILE *stream;
char *message;
int v;

void parse_prt(message,v)
char *message;
int v;

void parse_uprm(stream,message,modes,v)
FILE *stream;
char *modes,*message;
int v;

void parse_prm(message,modes,v)
char *modes,*message;
int v;

void parse_uwtext(stream,message)
FILE *stream;
char *message;

void parse_wtext(message)
char *message;

void parse_updt(stream,message)
FILE *stream;
char *message;
```

void parse_pdt(message)
char *message;

void parse_err(s1,s2)
char *s1,*s2;

void parse_er(s1)
char *s1;

void parse_uprc(stream,message,v)
FILE *stream;
char *message;
char v;

void parse_prc(message,v)
char *message;
char v;

void parse_rcsid_print(char *rcsid)

void parse_time_print(ofd,s)
FILE *ifd;
char *s;

void parse_time_init()

DESCRIPTION

These functions print information, **int**, **float**, etc., on **stdout**, **stderr**, or a specified stream. The information to be printed is contained or pointed to in the variable *v.* In most of the functions, a message, which is contained in *message,* is printed before the information. Unless otherwise noted, if the message begins with an asterisk, "*", the message with the asterisk deleted will be printed and the program will exit with a call to **exit**(3). No message will be printed if *message* is set to NULL.

parse_pri( ) and **parse_upri( )** print **int** on **stdout** and the specified stream, respectively.

parse_prf( ) and **parse_uprf( )** print **float** on **stdout** and the specified stream, respectively.

parse_prs( ) and **parse_uprs( )** print strings's on **stdout** and the specified stream, respectively.

parse_prt( ) and **parse_uprt( )** print toggle's on **stdout** and the specified stream, respectively. The value, *v,* can contain only zero, which corresponds to "no", or any other value, which corresponds to "yes". The message, if present, is appended with ": " before printing the toggle contained in *v* .

parse_prm( ) and **parse_uprm( )** print mode's on **stdout** and the specified stream, respectively. The character at address *modes+v* is printed. The message, if present, is appended with ": ".

parse_prc( ) and **parse_uprc( )** print **char** on **stdout** and the specified stream, respectively.

parse_wtext( ) and **parse_uwtext( )** print strings's on **stdout** and the specified stream, respectively. A newline, "\n", is printed after the text. A message beginning with an asterisk (*) will not cause the functions to exit.

parse_pdt( ) and **parse_updt( )** print the date and time on **stdout** and the specified stream, respectively. The message, if present, is appended with ": ".

parse_er( ) and **parse_err( )** print one and two strings, respectively, on **stderr**. No space is inserted between the two strings. The programs always exit with a call to **exit**(3).

parse_rcsid_print( ) prints on stdout the message "rcs revision:" followed by the revision number contained in the RCS identification string *Id.* The letter 'L' is appended if the file is locked.

parse_time_print( ) prints the user, system, and elapsed times since the last call to **parse_time_init( )** or **parse_time_print( ).** The string *s* is printed before the times if the string is not null. The output is sent to *ofd.* See the **man**(1) pages for **gettimeofday**(3) and **getrusage**(3) for definitions of user, system and elapsed times.

SEE ALSO

parse(3), **parse_accept**(3), **rcs**(1), **gettimeofday**(3), **getrusage**(3)

AUTHOR

Carl R. Crawford

NAME

parse_toupper, parse_tolower, parse_itoa, parse_words, parse_suffix, parse_suffix_malloc, parse_remove_suffix, parse_root, parse_host – miscellaneous string manipulation functions

SYNOPSIS

#include <parse.h>

void parse_toupper(s)
char *s;

void parse_tolower(s)
char *s;

void parse_words(s,argc,argv,mword)
char *s;
int *argc,mword;
char **argv;

char *parse_itoa(v,s)
int v;
char *s;

char *parse_suffix_malloc(file,suffix)
char *file;
char *suffix;

void parse_suffix(file,suffix)
char *file;
char *suffix;

void parse_remove_suffix(file)
char *file;

char * parse_extract_suffix(file)
char *file;

char *parse_root(file)
char *file;

char *parse_host()

DESCRIPTION

parse_toupper( ) converts the string contained in *s* to upper case.

parse_tolower( ) converts the string contained in *s* to lower case.

parse_words( ) finds the words contained in the string *s.* A word is defined to be the text surrounded by white space, where white consists of SPACE and TAB characters. The words contained in *s* are NULL terminated. The number of words contained in *s* is returned in *argc*. A list of pointers to the words is returned in *argv.* The *argv* array must be allocated by the user. The variable *mword* is the size of *argv.* It is an error to have more than *mword* words in the string. An error will be printed in this case on **stderr** and the program will be exited with a call to **exit**(3).

parse_itoa( ) converts the **int** contained in *v* to a string and returns the string in *s.* The maximum length of the string, including the terminating NULL character, is given by **PARSE_ITOA.** If *s* is NULL then space for the string will be allocated with **parse_malloc**(3). In all cases the address of the string, either *s* or the output of **parse_malloc**(3), is returned.

parse_suffix( ) appends, if necessary, a filename extension, *suffix,* to the the filename, *file.* Necessary means that a period (.) is not contained in the filename. Periods in the optional path that precedes the filename are ignored. The suffix should not contain a period as one is appended to the filename before the suffix is appended. The buffer pointed to by *file* must be long enough to contain the filename with the appended suffix. **parse_suffix_malloc( )** performs the same function with the exception that the resulting filename is placed into a new buffer that is acquired with a call to **parse_malloc**(3). Use **parse_free**(3) to

free the space occupied by the new filename.

parse_remove_suffix( ) removes a suffix, if present, in the filename contained in *file*. The deletion is done by setting the byte containing the first period (.) in the filename to zero. Periods in the optional path that precedes the filename are ignored.

parse_extract_suffix( ) extracts a suffix, if present, in the filename contained in *file*. The suffix is defined as the string following the first period (.) in the filename. Periods in the optional path that precedes the filename are ignored. The suffix is returned in a malloc'ed array. A NULL is returned if a suffix is not present.

parse_root( ) handles tildes (~) that might be present in a filename *file*. If the filename begins with "~/", then the filename will be prepended with the home directory found in the system **passwd**(5) file for the user of program. If the filename begins with *~user,* then the filename will be prepended with the home directory of *user*. The resulting filename is copied to a static buffer whose address is returned.

parse_host( ) returns the name of the host on which the program is executing.

SEE ALSO

parse(3), **parse_malloc**(3), **parse_free**(3), **getpwnam**(3), **getpwuid**(3), **passwd**(5),

AUTHOR

Carl R. Crawford

NAME

crcplot.a – graphics primitives library

SYNOPSIS

The CRC plotting package consists of the following functions:

plots( ) initializes the system for plotting. In particular, it opens an interprocess communication channel to a plotting window, usually **xplot**(1), on the local host. **plotss( ), plots_host( ),** and **plotss_host( )** are variants that allow control of error paths and the host on which the plotting window is running.

plot( ) draws lines and manages the current position and origin of the plotting coordinate system.

factor( ) sets new drawing factor.

where( ) returns current position and drawing factor.

symbol( ) plots character text and special symbols.

number( ) plots numeric values of program variables.

scale( ) computes and sets scale factors for data arrays.

line( ) plots arrays of data for line or centered-symbol graphs.

dline( ) plots arrays of data with dashed lines.

curve( ) plots arrays of data after fitting a smooth curve to it.

axis( ) draws axes including tick marks, scale annotations and axis labels.

grid( ) draws horizontal and/or vertical grid patterns.

plot3d( ) generates a line drawing of a three-dimensional surface at a specified orientation.

cplot( ) generates contours of iso-intensity for a two-dimensional function.

clear( ) clears the plotting window or starts a new page on the hardcopy device.

hatch( ) draw hatch marks in polygons.

hardcopy( )
tells the plotting window to generate a hardcopy of the displayed graphics.

mouse( ) blocks execution until the left mouse button is depressed in the display program; the coordinates of the mouse are are returned.

DESCRIPTION

The plotting package derives its roots from the days when all plotting was done on flat bed plotters. These plotters consisted of a pen attached to a two-dimensional positioning device and a piece of paper held in place under the pen using vacuum attachment to the bed of the plotter. The pen could be placed in either an up or down state that were also known as the move and draw positions, respectively. A computer could draw arbitrarily positioned line segments by first moving the pen to one end of the segment, dropping the pen, and then moving the pen to the other end of the line segment. Of course the speed of the positioners would have to be controlled in a way to draw a straight line with uniform density of deposited ink. The ability to draw lines could be built upon to draw more complex graphical objects like characters, axes and smooth curves.

Flat bed plotters are no longer the principal plotting devices. They have given way to high resolution video displays, laser printers, electrostatic printers, film recorders, and graphic formats such as Postscript. In order to remain backwards compatible with software that drives flat bed plotters, this package retains the user interface and nomenclature of a flat bed plotter. However, as is explained below, the resulting graphics output is sent to the newer devices. For the sake of illustration, the library will be presented as if the plotters were still in use. The extensions required to support the newer plotting peripherals are explained below.

The main routines in the graphics library are **plots( )** and **plot( ).** The user is required to call **plots( )** first in order to connect to the plotter and initialize the plotting package. The function **plot( )** controls where the

pen is located and whether the pen is its up or down state. Finally, the user has to detach the plotter using **plot( )** with a special argument in place of the up/down flag. Higher order objects, such as axes and text, are drawn with other routines such as **axis( )** and **symbol( ).** However, the code to draw the higher order objects always reduces to calling **plot( ).**

The syntax of **plot( )** is:

```
plot(x,y,ipen)
float x,y
int pen
```

where *(x,y)* is the location to which to move the pen. The units and the origin of the plotting system are discussed below. The *ipen* argument determines the state of the pen where the values two and three correspond to down and up, respectively. The plotter is detached when *ipen* is 999. As an example, the complete code to draw a line from (2,2) to (10,3) is:

```
plots();
plot(2.0,2.0,2);
plot(10.0,3.0,3);
plot(0.,0.,999);
```

The first call to **plot( )** moves the pen to (2,2) and the second draws the line from (2,2) to (10,3). The last call detaches the plotter.

The concept of a "current point" or "current position" is intrinsic to **plot( ).** The current point is the argument, *(x,y),* of the previous call to **plot( ).** In terms of the flat bed plotter, it is where a call to **plot( )** leaves the pen at the end of the call. **plot( )** draws lines with respect to the current point. Therefore, an arbitrarily positioned line segment has to be drawn with two calls to **plot( ).** When drawing lines that share endpoints, it is not necessary to move to the beginning of a segment because the end of one segment is the start of another. Consider drawing a box with sides five units long with its lower-left corner located at (4,6). The code to do this could be:

```
plot(4.,6.,3);
plot(4.+5.0,6.,2);
plot(4.+5.0,6.+5.0,2);
plot(4.,6.+5.0,2);
plot(4.,6.,2);
```

(The call to **plots( )** and the last call to **plot( )** with *ipen* set to 999 have not been included for the sake of brevity.) The first call moves to the lower left corner; the second draws the bottom of the box; the third draws the right side; the fourth draws the top; and last call draws the left side.

By default, all coordinates are given in units of inches with respect to the plotter's origin, usually the lower-left corner of paper. The plotting scale can be converted to another unit by telling **plot( )** to multiply all of its given coordinates by a specified value before plotting in inches. The value is specified with **factor( ).** For example, if one wants to plot in centimeters instead of inches, one should use

```
factor(1/2.54)
```

after calling **plots( )** but before calling **plot( ).**

The physical plotting origin can also be problematic. Consider the case where you have a subroutine that draws a complicated object by calling the other intrinsic functions in the package, say the box described above. Now you want to draw these boxes at various places on the page. One could pass the location of the box to the subroutine and have it add the location to all the calls to the intrinsic functions. Such a subroutine could be written as:

```
box(x,y)
float x,y;
{
        plot(x,y,3);
        plot(x+5.0,y,2);
```

```
        plot(x+5.0,y+5.0,2);
        plot(x,y+5.0,2);
        plot(x,y,2);
}
```

To make the life of the programmer easier, a method is included to change the origin to any place on the page. If the state of the pen is negative (normally it is positive) when calling **plot( ),** the coordinates passed to the function become the new plotting origin. All plotting done with subsequent calls to **plot( )** is done with respect to the new origin. Therefore, all the follow examples draw lines from (3,3) to (7,8):

```
plot(3.0,3.0,3);
plot(7.0,8.0,2);
```

or

```
plot(5.0,5.0,-3);
plot(-2.0,-2.0,3);
plot(2.0,3.0,2);
```

or

```
plot(3.0,3.0,-3);
plot(4.0,5.0,2);
```

In the case of the box example, you only have to set the origin before each call to the box function. The box function then does not have to know anything about where on the page it is plotting, only how it is plotting with respect to the origin. Of course, the scaling functionality available with **factor( )** could be combined to draw bigger or smaller boxes at different positions on the page. Boxes of with sides that are four inches long can be drawn *centered* at (4,3) and (8,2) as follows:

```
plot(4.0,3.0,-3);
boxc();
plot(4.0,-1.0,-3);
boxc();
```

where **boxc( )** is given by

```
boxc(){
        plots(-2.0,-2.0,3);
        plots(2.0,-2.0,2);
        plots(2.0,2.0,2);
        plots(-2.0,2.0,2);
        plots(-2.0,-2.0,2);
}
```

Notice that the second box, which should be drawn at (8,2), does not use

```
plots(8.0,2.0,-3);
```

This is because all pen movement is with respect to the prevailing origin. The origin before this call is already at (4,3). Therefore, to reorigin at (8,2), the argument is formed from the difference of (8,2) and (4,3) yielding (4,-1).

The use of a negative argument in the call to **plot( )** illustrates another feature (maybe it is a problem) of the library. Some of the function arguments are used for multiple purposes instead of having functions with myriad arguments. The multifaceted nature is usually obtained by using an argument's magnitude for one parameter and its sign for another purpose, albeit binary.

This ends the description of the basic features of the library. Please refer to the **man**(1) pages for complete information on the functions listed above. The rest of this section discusses how the library is extended to talk to output devices other than flat bed plotters.

The UNIX operating system employs pipes to use commands as building blocks to form more complicated functions. In order for programs to communicate through pipes, they must agree on common format for the

data they will exchange. The exchange format for graphics is **plot**(5). A library of functions exist in the **plot**(3) library to generate **plot**(5) commands. Programs, called filters, exist to display the resulting graphics directly on video displays or to convert it into formats understood by other devices. For example, the filter **plotps**(1) converts **plot**(5) to Postscript. The **crcplot**(3) library uses a slightly modified version of the **plot**(3) library (the differences are explained below) to generate **plot**(5) format. Therefore, the user of **crcplot** can get to any device if the appropriate filters exist.

A design goal of this library was to get graphics displayed, by default, on a video display. Because of the proliferation of windowing systems, the portion on the video display in which the plot is drawn is referred to as the plotting window. Malcolm Slaney wrote two programs that read **plot**(5) commands [from standard input so the programs are filters] and display the graphics in a window. The program is **xplot**(1) for the X windowing system.

Slaney's filters have two drawbacks. The first is that the user has to specify the necessary shell constructs to get the graphics to the filter. Binary information would be sent to the terminal if the user would forget to pipe the application program to the filter. The second drawback is that the window exists only as long as the application program is running. For some applications this is not too serious. However, there is no easy way to have several programs write to the same plotting window.

Slaney's filters were modified to receive graphic commands through interprocess communication instead through a pipe. Specifically, the communication is made over sockets, which, for the those not familiar with them, are basically telephone times with corresponding telephone numbers. The application code makes a socket connection (by dialing the appropriate telephone number) to the modified plotting windows and then writes the graphics to the filter via the socket. The key here is that the plotting window has to be running before the application program is run. This mode of operation guarantees that the graphics remain visible after the application program has finished. Also, more than one graphics application can write to the plotting window so that composite pictures can be generated.

The **plot**(3) library was modified so that the call that initialized plotting, **openpl**(3), automatically connects to the socket attached to the plotting window. In practice, a new **plot**(3) was written because the source code for the original was unavailable. The new library is called **plotcrc**(3) and is found is found in **crcplot.a**(3). In order to distinguish the new library from the original, all the command names are prefixed with *plot_*. For example, **openpl( )** was renamed **plot_openpl( ).**

By default, the connection is made to the plotting window on the local host, the host on which the application program is running. The user can specify the name of the machine on which the plotting window is running by setting the environment variable **XPLOTHOST.** The user's program will be forced to exit if the connection cannot be made. Variants of the initialization function exist for connecting to plotting windows on other hosts and to return to the called program instead of exiting. The output of an application program can be sent to standard output if the host is set to the string *stdout* either with a function call or with the environment.

As indicated above, the **crcplot** library calls the modified **plot**(3) library **plotcrc**(3). In **crcplot, plots( )** is basically a call to **plot_openpl( ).** Therefore, when one calls **plots( )** the graphics output is sent directly to the plotting window.

Because plotting windows can be different sizes, it can be debated how to specify the units of distance within the window. Clearly the window has a size if terms of number of pixels. However, there is no capability for the plotting window to send its size back to the application program. Even if it could, the user might have to tailor application code to the size. The method that is employed in the package is to assume that the smallest dimension of the plotting window, its width or height, corresponds to ten inches. (Sometimes inches are just called plotting units.) The other dimension is found by multiplying or dividing by the aspect ratio of the plotting window. The plotting window automatically handles the scaling from inches to pixels. Aspect ratios of up to about two to one can be used.

FILES

libcrcplot.a

actual name of the library **crcplot.a.** The library can be accessed with **-lcrcplot** when using a C compiler or the linker.

ENVIRONMENT

XPLOTHOST

Can be used to specify the host and the socket for **plots( ), plotss( ),** and **plot_openpl( ).** It has the form *[host][:socket].*

INTERPROCESS COMMUNICATION

8124 Default socket used to talk to plotting window.

SEE ALSO

cplot(1), **plot3d**(1), **plotps**(1), **xpc**(1), **xpic**(1), **xplot**(1), **axis**(3), **clear**(3), **cplot**(3), **curve**(3), **dline**(3), **factor**(3), **grid**(3), **hardcopy**(3), **hatch**(3), **line**(3), **mouse**(3), **number**(3), **plot**(3), **plot**(3x), **plotcrc**(3), **plot3d**(3), **plots**(3), **scale**(3), **symbol**(3), **where**(3), **plot**(5),

AUTHOR

Carl R. Crawford

BUGS

For the sake of speed, lines are not clipped. Therefore, lines drawn outside the visible plotting area can alias back into the display.

There is no way to control the absolute size of the final hardcopy. The user has to develop appropriate scaling factors on a system-by-system basis.

NAME

axis – generates a positioned and annotated axis

SYNOPSIS

#include <crcplot.h>

void axis(x,y,label,nchar,axlen,angle,fval,dv)
float *x,*y;
double axlen,angle,fval,dv;
int nchar;
char *label;

DESCRIPTION

axis( ) may be used to generate a positioned axis line with labels, scale annotations, and tick marks.

ARGUMENTS

x,y defines the starting coordinates for the axis line.

label is an alphanumeric text string which will be centered and used to label the generated axis.

nchar defines the number of characters (absolute) in the alphanumeric text, *label*. If *nchar* is set to 999 or -999, then the length of the string will be calculated. This feature only works if the string is terminated with a zero.

+*nchar* indicates that all labeling, annotations, and tick marks are to be generated above the axis line (that is, counterclockwise) as is normally used for the y-axis.

-*nchar* indicates that all labeling, annotations, and tick marks are to be generated below the axis line (that is, clockwise) as is normally used for the x-axis.

axlen (absolute) is the length of the axis line in *units.* The minimum length is one.

+*axlen* indicates that the numerical annotation at each tick mark is offset uniformly. The offset is to the left for a horizontal axis.

-*axlen* indicates that the numerical annotation at each tick mark is offset nonuniformly. Annotation is guaranteed not to extend past the linear extent of the axis. The mode is useful when plotting abutting axes that subtend angles other than 90 degrees.

angle is the angle, in degrees, at which the axis is to be drawn. Normally, zero degrees is used for generating the x-axis; Ninety degrees is used for generating the y-axis.

fval is the first value (either minimum or maximum) which will be used for annotating the axis at *unit* (for example: inch) intervals.

dv represents the number of data units per axis *unit*. This 'delta' value will be added to the *fval* starting value for generating scale annotations at each succeeding interval along the axis line.

SEE ALSO

cplot(1), **plot3d**(1), **plotps**(1), **xpc**(1), **xpic**(1), **xplot**(1), **axis**(3), **crcplot**(3), **clear**(3), **curve**(3), **dline**(3), **factor**(3), **grid**(3), **hardcopy**(3), **hatch**(3), **line**(3), **mouse**(3), **number**(3), **plot**(3), **plotcrc**(3), **plots**(3), **scale**(3), **symbol**(3), **where**(3)

AUTHOR

Carl R. Crawford

NAME

clear – erase the "screen"

SYNOPSIS

#include <crcplot.h>

void clear()

DESCRIPTION

clear( ) sends an erase command to the "plotter". If the output is directed to **xplot**(1), then its plotting canvas will be cleared. If the output is sent to the plotter directly with **lpr**(1), then a form-feed will be issued.

SEE ALSO

cplot(1), **plot3d**(1), **plotps**(1), **xpc**(1), **xpic**(1), **xplot**(1), **axis**(3), **crcplot**(3), **clear**(3), **curve**(3), **dline**(3), **factor**(3), **grid**(3), **hardcopy**(3), **hatch**(3), **line**(3), **mouse**(3), **number**(3), **plot**(3), **plotcrc**(3), **plots**(3), **scale**(3), **symbol**(3), **where**(3)

AUTHOR

Carl R. Crawford

NAME

cplot – generates contours of iso-intensity for a two-dimensional function

SYNOPSIS

#include <crcplot.h>

void cplot(x,y,z,nx,ny,xmin,xmax,ymin,ymax,xlen,ylen,pxaxis,pyaxis,
tv,nc,nix,niy,bl,tl,xl,yl)
float *x,*y,*z,*tv
double xmin,xmax,ymin,ymax,xlen,ylen
int nx,ny,nc,nix,niy,pxaxis,pyaxis
char *bl,*tl,*xl,*yl

DESCRIPTION

cplot( ) is used to generate contours of iso-intensity through a two-dimensional function. The function is defined by *z(x,y)*. Vectors corresponding to *x* and *y* are required along with an array corresponding to *z*. Minimum and maximum bounds are required for the two vectors. The data cannot be outside these bounds. However, the bounds can be wider than the actual data. The function can be magnified using linear interpolation to generate smoother contours.

The axis lengths are used to scale the data into the plotting space. The point at *(xmin,ymin)* is put at the current plotting origin.

A number of error messages are generated on standard output. The specific messages are listed below. In almost all cases, the errors are due to improper specification of the arguments. The function will exit with an error code of one in case of an error.

ARGUMENTS

x is an array containing the x locations of the function. The vector must be monotonically increasing and bounded by *xmin* from below and *xmax* from above.

y is an array containing the y locations of the function. The vector must be monotonically increasing and bounded by *ymin* from below and *ymax* from above.

z is an array corresponding to the function to have contours drawn through it. The size of the array is *nx* by *ny*. The x-index varies the fastest.

nx is the size of the *x* vector.

ny is the size of the *y* vector.

xmin bounds the *x* vector from below.

xmax bounds the *x* vector from above.

ymin bounds the *y* vector from below.

ymax bounds the *y* vector from above.

xlen,ylen are the lengths of the two axes. These values are used for scaling the data.

pxaxis,pyaxis
A non-zero value indicates that the specified axis should be plotted.

tv is a vector of values for which contours are to be drawn.

nc is the size of the *tv* vector.

nix,niy are the linear interpolation factors. The values are actually the number of extra points interpolated between given samples. Therefore, a value of zero indicates no interpolation and a value of one is for magnification by two. In general the magnification factor is one plus the specified value.

tl is a text string to be plotted on top (above) of the plot. The text is centered.

bl is a text string to be plotted below the plot. The text is centered.

xl,yl are text strings to be plotted next to the specified axes.

SEE ALSO

cplot(1), **cplot**(1), **plotps**(1), **xpc**(1), **xpic**(1), **xplot**(1), **axis**(3), **crcplot**(3), **clear**(3), **cplot**(3), **curve**(3), **dline**(3), **factor**(3), **grid**(3), **hardcopy**(3), **hatch**(3), **line**(3), **mouse**(3), **number**(3), **plot**(3), **plotcrc**(3), **plot3d**(3), **plots**(3), **scale**(3), **symbol**(3), **where**(3)

AUTHOR

Carl R. Crawford

DIAGNOSTICS

cplot: nx and ny must be greater than one
The *x* and *y* vectors must have at least two points in each.

cplot: [xy] data out of range
The data in the vectors are not bounded by their *min* and *max* values.

cplot: no [xy] range
The *min* and *max* values are equal for a vector.

cplot: improper axis length
One or more of the axes lengths is less than or equal to zero.

cplot: malloc failure
The function cannot allocate temporary space for working vectors.

cplot: no contours
There has to be at least one entry in *tv*.

cplot: illegal interpolation factors
The interpolation factors cannot be negative.

NAME

curve – generate smooth line through points

SYNOPSIS

#include <crcplot.h>

void curve(x,y,ne,delta)
float *x,*y;
double delta;
int ne;

DESCRIPTION

curve( ) is used to generate a smooth continuous curved line through a series of user-defined coordinate data points. The coordinate data points may be scaled or unscaled, and the generated line may optionally be dashed or solid. The algorithm employed is invariant under axis rotation and uses local procedures for incrementally approximating a smooth curve.

At the minimum three points are required for curve approximation. If *ne* is specified as two the two data points will be joined with a straight line. If *ne* is specified as zero or one; or *delta* is defined as zero no plotting takes place and execution returns directly to the calling program.

Scaled coordinate data values are computed (1<i<*ne*):

xx = (x(i)-x(|ne|+1))/x(|ne|+2)

yy = (y(i)-y(|ne|+1))/y(|ne|+2)

ARGUMENTS

x,y are arrays of coordinate data points to be joined by a smooth curve.

ne (absolute) is the number of coordinate points in *x* and *y*. A negative value of *ne* indicates that scale factors are located as the last two elements of each data array (see subroutine **scale**(3)). A positive value of *ne* indicates that the coordinate points are already scaled for plotting (no scale factors). (Note - Only a subroutine **scale**(3) increment argument of one can be used when computing scale factors for this function.)

delta (absolute) is the segment length for the incremental approximation of the curve.

+*delta* indicates that the curve is to be generated with a solid line.

-*delta* indicates that the curve is to be generated with dashed lines (of |*delta*|**length*).

SEE ALSO

cplot(1), **plot3d**(1), **plotps**(1), **xpc**(1), **xpic**(1), **xplot**(1), **axis**(3), **crcplot**(3), **clear**(3), **curve**(3), **dline**(3), **factor**(3), **grid**(3), **hardcopy**(3), **hatch**(3), **line**(3), **mouse**(3), **number**(3), **plot**(3), **plotcrc**(3), **plots**(3), **scale**(3), **symbol**(3), **where**(3)

AUTHOR

Carl R. Crawford

NAME

dline – plot points from data arrays with dashed lines

SYNOPSIS

#include <crcplot.h>

void dline(x,y,npts,dsh,gap,m)
float *x,*y,*dsh,*gap;
int npts,m;

DESCRIPTION

dline( ) may be used to plot points from coordinate data arrays with dashed lines. Scaling parameters (*fval* and *dv* as set by subroutine **scale**(3)) must immediately follow each array.

Arbitrarily complex dash-gap sequences may be specified. The function first plots a dash of length *dsh[0],* then a gap of length *gap[0],* then a dash of length *dsh[1],* etc. After plotting *gap[m-1],* the sequence is repeated starting again with *dsh[0].*

ARGUMENTS

x is an array containing abscissa (x) values with scaling parameters for the x-array.

y is an array containing ordinate (y) values with scaling parameters for the y-array.

npts is the number of data values in each of the two, *x,y* arrays. This number does not include the extra two locations containing the scaling parameters.

dsh a pointer to an array of length *m* containing the lengths of the sequence of dashes to be used in plotting the relationship.

gap a pointer to an array of length *m* containing the lengths of the sequence of spaces between dashes to be used in plotting the relationship.

m the number of dashes and gaps in the dash-gap sequence.

SEE ALSO

cplot(1), **plot3d**(1), **plotps**(1), **xpc**(1), **xpic**(1), **xplot**(1), **axis**(3), **crcplot**(3), **clear**(3), **curve**(3), **dline**(3), **factor**(3), **grid**(3), **hardcopy**(3), **hatch**(3), **line**(3), **mouse**(3), **number**(3), **plot**(3), **plotcrc**(3), **plots**(3), **scale**(3), **symbol**(3), **where**(3)

AUTHOR

Carl R. Crawford

NAME

factor – enlarge or reduce the plot size

SYNOPSIS

#include <crcplot.h>

void factor(fact)
double fact;

DESCRIPTION

factor( ) allows the programmer to enlarge or reduce the size of the entire plot by specifying the ratio of the desired plot size to the normal plot size.

ARGUMENTS

fact ratio of desired size to "normal" size (for example: *fact*=2 gives a double size plot, *fact*=0.5 gives a half size plot.

SEE ALSO

cplot(1), **plot3d**(1), **plotps**(1), **xpc**(1), **xpic**(1), **xplot**(1), **axis**(3), **crcplot**(3), **clear**(3), **curve**(3), **dline**(3), **factor**(3), **grid**(3), **hardcopy**(3), **hatch**(3), **line**(3), **mouse**(3), **number**(3), **plot**(3), **plotcrc**(3), **plots**(3), **scale**(3), **symbol**(3), **where**(3)

AUTHOR

Carl R. Crawford

NAME

grid – generate grid and overlay patterns

SYNOPSIS

#include <crcplot.h>

void grid(x,y,nx,xd,ny,yd)
int nx,ny;
float *xd,*yd;
double x,y;

DESCRIPTION

grid( ) allows the programmer to generate a variety of grid patterns or overlay forms. The user indicates the spacing values between horizontal and vertical lines and the number of intervals desired in each direction. The design features incorporated in this function facilitate the generation of special uniform, semilog, log-log, exponential, or other user defined grid formats.

ARGUMENTS

x,y are the starting coordinates (lower left-hand corner) of the grid to be generated.

nx the number of intervals in the x direction. If *nx* is greater than 1000, argument *xd* will be treated as an array of interval values with *nx*-1000 elements. *-nx* indicates that the actual vertical line generations are to be suppressed. Note that one more grid line is generated than the number of intervals specified by *nx.* Also note that x-intervals are bounded by vertical (y-direction) lines.

xd is the distance between uniformly spaced vertical lines (*nx*<l000), or an array of values for spacing vertical lines at varying intervals (*n*x>1000).

ny is the number of intervals in the y-direction. If *ny* is greater than l000, argument *yd* will be treated as an array of interval values with *ny*-1000 elements. *-ny* indicates that the actual horizontal line generations are to be suppressed. Note that one more grid line is generated than the number of intervals specified by *ny.* Also note that y-intervals are bounded by horizontal (x-direction) lines.

yd is the distance between uniformly spaced horizontal lines (*ny*<1000), or an array of values for spacing horizontal lines at varying intervals (*ny*>1000).

SEE ALSO

cplot(1), **plot3d**(1), **plotps**(1), **xpc**(1), **xpic**(1), **xplot**(1), **axis**(3), **crcplot**(3), **clear**(3), **curve**(3), **dline**(3), **factor**(3), **grid**(3), **hardcopy**(3), **hatch**(3), **line**(3), **mouse**(3), **number**(3), **plot**(3), **plotcrc**(3), **plots**(3), **scale**(3), **symbol**(3), **where**(3)

AUTHOR

Carl R. Crawford

NAME

hardcopy – invoke the "Hard" button on the xplot panel

SYNOPSIS

#include <crcplot.h>

void hardcopy()

DESCRIPTION

hardcopy( ) invokes the "Hard" button on the **xplot**(1) panel.

SEE ALSO

cplot(1), **plot3d**(1), **plotps**(1), **xpc**(1), **xpic**(1), **xplot**(1), **axis**(3), **crcplot**(3), **clear**(3), **curve**(3), **dline**(3), **factor**(3), **grid**(3), **hardcopy**(3), **hatch**(3), **line**(3), **mouse**(3), **number**(3), **plot**(3), **plotcrc**(3), **plots**(3), **scale**(3), **symbol**(3), **where**(3)

AUTHOR

Carl R. Crawford

BUGS

The routine was implemented using a new **plot**(5) command called 'h'. Only **xplot**(1) recognizes this command. Other plot filters may complain about its presence.

NAME

hatch – draw hatch marks in polygon

SYNOPSIS

#include <crcplot.h>

void hatch(x,y,n,ang,pitch)
float *x,*y;
double angle,pitch;
int n;

DESCRIPTION

hatch( ) may be used to draw hatch marks inside of a polygon. The last vertex of the polygon is assumed to be connected to first vertex. The polygon can be convex, can contain edges that intersect and can contain overlapping vertices. The last feature allows for holes in filled polygons.

ARGUMENTS

x is an array containing abscissa (x) locations of the vertices of the polygon.

y is an array containing ordinate (y) locations of the vertices of the polygon.

n (absolute) is the number of values in each of the two, *x,y* arrays. Recall that the an edge is assumed from the last vertex back to the first vertex.

+n indicates that the edges of the polygon should be drawn.

-n indicates that the edges of the polygon should not be drawn.

angle is the angle in degrees that the hatch marks make respect to the x-axis.

pitch is the distance in plot units between the hatch lines.

SEE ALSO

cplot(1), **plot3d**(1), **plotps**(1), **xpc**(1), **xpic**(1), **xplot**(1), **axis**(3), **crcplot**(3), **clear**(3), **curve**(3), **dline**(3), **factor**(3), **grid**(3), **hardcopy**(3), **hatch**(3), **line**(3), **mouse**(3), **number**(3), **plot**(3), **plotcrc**(3), **plots**(3), **scale**(3), **symbol**(3), **where**(3)

AUTHOR

Carl R. Crawford

NAME

line – plot points from coordinate data arrays

SYNOPSIS

#include <crcplot.h>

void line(x,y,npts,inc,lintyp,inteq)
float *x,*y;
int npts,inc,lintyp,inteq;

DESCRIPTION

line( ) may be used to plot points from coordinate data arrays. Data values may be represented by centered symbols and/or connecting lines. Scaling parameters (*fval* and *dv* as set by subroutine **scale**(3)) must immediately follow each array.

ARGUMENTS

x is an array containing abscissa (x) values with scaling parameters for the x-array.

y is an array containing ordinate (y) values with scaling parameters for the y-array.

npts is the number of data values in each of the two, *x,y* arrays. This number does not include values skipped by *inc* nor the extra two locations containing the scaling parameters.

inc is the increment used in gathering data values from the two arrays (for example: *inc*=l will index each data element, *inc*=2 will index every other data element, *inc*=3 will index every third data element, and so forth).

lintyp defines the line type/symbol to be used by subroutine *line.* If symbols are plotted, the magnitude of *lintyp* (absolute) determines the symbol plot frequency; (for example: *lintyp*=+4 indicates that a special symbol is to be plotted at every fourth data point).

- *lintyp*=0 indicates that data points are to be connected by straight lines only; no symbols plotted.
- +*lintyp* indicates that data points are to be connected by straight lines in conjunction with plotting symbols.
- -*lintyp* indicates that no lines are to be drawn; only the symbols plotted.

inteq is an integer equivalent (that is, symbol number) defining the character/symbol to be used when symbol plotting (that is, *lintyp*≠0).

SEE ALSO

cplot(1), **plot3d**(1), **plotps**(1), **xpc**(1), **xpic**(1), **xplot**(1), **axis**(3), **crcplot**(3), **clear**(3), **curve**(3), **dline**(3), **factor**(3), **grid**(3), **hardcopy**(3), **hatch**(3), **line**(3), **mouse**(3), **number**(3), **plot**(3), **plotcrc**(3), **plots**(3), **scale**(3), **symbol**(3), **where**(3)

AUTHOR

Carl R. Crawford

NAME

mouse – return position of mouse in plot coordinate system

SYNOPSIS

#include <crcplot.h>

void mouse(x,y)
float *x,*y;

DESCRIPTION

mouse( ) blocks (prevents further execution) until the left mouse button is depressed in the main plotting window of the display program. The coordinates of the mouse are then returned in the calling arguments. The mouse coordinates are given in plotting units - not in the coordinates of the display program - suitable for using as arguments to other plotting routines. It is recommend that you notify the user to position and press the mouse button before calling this function. An error is generated if the function is called when a display program is not attached.

ARGUMENTS

x,y variables to be filled with the coordinates of the present mouse position.

SEE ALSO

cplot(1), **plot3d**(1), **plotps**(1), **xpc**(1), **xpic**(1), **xplot**(1), **axis**(3), **crcplot**(3), **clear**(3), **curve**(3), **dline**(3), **factor**(3), **grid**(3), **hardcopy**(3), **hatch**(3), **line**(3), **mouse**(3), **number**(3), **plot**(3), **plotcrc**(3), **plots**(3), **scale**(3), **symbol**(3), **where**(3)

AUTHOR

Carl R. Crawford

NAME

number – draw floating point numbers on user's graph

SYNOPSIS

#include <crcplot.h>

void number(x,y,height,fpn,angle,ndec)
int ndec;
double x,y,height,fpn,angle;

DESCRIPTION

number( ) may be used to convert a floating-point number to the appropriate decimal equivalent which is then drawn on the user's plot.

ARGUMENTS

x,y is the starting coordinate defining the lower left-hand corner of the first character to be produced. If x and/or y are equal to 999.0 annotation is continued from the current coordinate position (See **symbol**(3)).

height defines the character height, in *units.*

fpn is the floating point number to be plotted

angle is the angle, in degrees measured from the x-axis, at which the numeric annotation is plotted (that is, the vector direction of the text string).

ndec specifies the number of decimal digits which are to be plotted:

- +*ndec* defines the number of digits to the right of the decimal point which are to be plotted, after rounding.
- *ndec*=0 indicates that only the integer position and decimal point are to be plotted, after rounding.
- *ndec*=-1 indicates that only the integer portion is to be plotted, after rounding (no decimal point).
- *ndec*<-1 specifies that |*ndec*|-1 digits are to be right truncated from the integer portion and plotted after rounding.

SEE ALSO

cplot(1), **plot3d**(1), **plotps**(1), **xpc**(1), **xpic**(1), **xplot**(1), **axis**(3), **crcplot**(3), **clear**(3), **curve**(3), **dline**(3), **factor**(3), **grid**(3), **hardcopy**(3), **hatch**(3), **line**(3), **mouse**(3), **number**(3), **plot**(3), **plotcrc**(3), **plots**(3), **scale**(3), **symbol**(3), **where**(3)

AUTHOR

Carl R. Crawford

NAME

plot – draw and move straight lines

SYNOPSIS

#include <crcplot.h>

void plot(x,y,ipen)
double x,y;
int ipen;

DESCRIPTION

plot( ) processes straight line "pen" moves with the pen either up or down during movement.

ARGUMENTS

x,y is a single coordinate-pair defining the terminal position of the "pen" move. The line is drawn from the terminal position of the previous call to this function. The coordinate pair is given in an absolute coordinate system. The origin of the space can be controlled using negative values of *ipen*.

ipen is a signed integer value controlling the "pen" up/down status, re-origin, offset/scale, and end of plot requests.

ipen=999	End of plot.
ipen=+3	Move to *(x,y).*
ipen=+2	Draw to *(x,y).*
ipen=-2	Draw to *(x,y),* re-origin.
ipen=-3	Move to *(x,y),* re-origin.

SEE ALSO

cplot(1), **plot3d**(1), **plotps**(1), **xpc**(1), **xpic**(1), **xplot**(1), **axis**(3), **crcplot**(3), **clear**(3), **curve**(3), **dline**(3), **factor**(3), **grid**(3), **hardcopy**(3), **hatch**(3), **line**(3), **mouse**(3), **number**(3), **plot**(3), **plotcrc**(3), **plots**(3), **scale**(3), **symbol**(3), **where**(3)

AUTHOR

Carl R. Crawford

NAME

plot_openpl, plot_openpl_fd, plot_openpl_host, plot_erase, plot_label, plot_line, plot_circle, plot_arc, plot_move, plot_cont, plot_point, plot_linemod, plot_space, plot_closepl, plot_host, plot_port, plot_fd, plot_hardcopy, plot_mouse – graphics interface to plotting windows using **plot**(5) format

SYNOPSIS

```
#include <stdio.h>
#include <crcplot.h>

int plot_openpl( )

void plot_openpl_fd(fd)
FILE *fd;

int plot_openpl_host(host)
char *host;

void plot_erase( )

void plot_label(s)
char *s;

void plot_line(x1, y1, x2, y2)
int x1, y1, x2, y2;

void plot_circle(x, y, r)
int x, y, r;

void plot_arc(x, y, x0, y0, x1, y1)
int x, y, x0, y0, x1, y1;

void plot_move(x, y)
int x, y;

void plot_cont(x, y)
int x, y;

void plot_point(x, y)
int x, y;

void plot_linemod(s)
char *s;

void plot_space(x0, y0, x1, y1)
int x0, y0, x1, y1;

void plot_closepl( )

char *plot_host( )

int plot_port( )

FILE *plot_fd( )

void plot_hardcopy( )

void plot_mouse(x , y)
int x, y;
```

DESCRIPTION

These subroutines generate graphic output in a relatively device-independent manner according to the **plot**(5) format using the interface standard set forth in **plot**(3x). The routines are called by the graphic primitives in **crcplot**(3). Normally, the user will not have to call these subroutines directly. The display program **xplot**(1) reads **plot**(5) commands. Two functions make connections to one of these programs, called plotting windows, via a socket connection. A graphics output is a stream of plotting instructions. Each instruction consists of an ASCII letter usually followed by bytes of binary information. The instructions are executed in order. A point is designated by four bytes representing the *x* and *y* values; each value

is a signed integer.

plot_openpl( ) or **plot_openpl_host( )** and then **plot_space( )** must be used before any of the others to open the device for writing. **plot_closepl( )** flushes the output. The last designated point in a **plot_line( ), plot_move( ), plot_cont( ),** or **plot_point( ),** call becomes the ''current point'' for the next instruction.

plot_openpl( ) opens a communication channel via sockets to a plotting window. The plotting window must be running before this function is called. By default the plotting window running on the local host is called. The default socket number used for interprocess communication is listed below in the section INTERPROCESS COMMUNICATION. One might want to override the host to provide a poor-person's X windowing system. One also might want to override the socket to run multiple copies of the plotting window on the same host. The environment can be used to do this overriding. **XPLOTHOST** is examined for the name of the host and the port number. The host and the socket are specified in a string of the form *[host][:socket]* where *host* is the name of the host and *socket* is the socket number. Either or both of the parts can be missing. Note that the colon (:) is attached to the socket number. If the host is missing, then the local host is used. If the socket is missing, then then default socket listed below is used. If host is set to the string *stdout* then the output is sent to *stdout* instead via the socket to the plotting window. A value of -1 will be returned if the connection cannot be made to the plotting window.

plot_openpl_host( ) is similar to **plot_openpl( )** with the exception that a connection is attempted to the plotting window running on the host named *host*. Note that the socket can be specified by appending the string *:socket* to the name of the host. The environment is not examined in this function.

plot_host( ) and **plot_port( )** returns the name of the host and port number (socket), respectively, used by **plot_openpl( )**. or **plot_openpl_host( )**.

plot_openpl_fd( ) specifies that the **plot**(5) commands should be written to the file descriptor *fd* .

plot_fd( ) returns the file descriptor corresponding to the communication channel to the plotting window. The command is provided in case **plot**(5) information needs to be communicated without the overhead of the subroutines described herein.

plot_closepl( ) flushes the graphics output and waits for a signal indicating plot completion from the plotting window.

plot_move( ) sets the current point to *(x,y)*.

plot_cont( ) draw a line from the current point to *(x,y)*. The current point is set to *(x,y)*.

plot_point( ) plots a point at *(x,y)*. The current point is set to *(x,y)*.

plot_line( ) draw a line from *(x0,y0)* to *(x1,y1)* . The current point is set to *(x1,y1)*.

plot_label( ) draw the ASCII string contained in *s* so that its first character falls on the current point. The argument is null-terminated and cannot contain NEWLINE characters.

plot_arc( ) draws a circular arc. The center is given by *(x,y)*, and the starting and ending points are given by *(x0,y0)* and *(x1,x1)*, respectively. The least significant coordinate of the end point is used only to determine the quadrant. The arc is drawn counter-clockwise.

plot_circle( ) draws a circle. The center of the circle is given by *(x,y)* and its radius is given by *r*.

plot_erase( ) start another frame of output by erasing the screen.

plot_linemod( ) specifies the style for drawing further lines. Valid values for *s* are ''dotted,'' ''solid,'' ''longdashed,'' ''shortdashed,'' and ''dotdashed.'' The argument is null-terminated and cannot contain NEWLINE characters.

plot_space( ) specifies the plotting area. The lower left corner of the plotting area is given by *(x0,y0)* and the upper right corner is given by *(x1,y1)*. The plot will be magnified or reduced to fit the plotting window as closely as possible. In every case the plotting area is taken to be square; points outside may be displayable on windows whose face is not square.

plot_hardcopy( ) invoke the "Hard" button on the plotting window. This command is not part of the normal **plot**(5) protocol.

plot_mouse( ) blocks (prevents further execution) until the left mouse button is depressed in the main plotting window of the display program. The coordinates of the mouse are then returned in the calling arguments. The mouse coordinates are given in plotting units - not in the coordinates of the display program - suitable for using as arguments to other routines listed in this section, It is recommend that you notify the user to position and press the mouse button before calling this function. An error is generated if the function is called when a display program is not attached. This command is not part of the normal **plot**(5) protocol.

ENVIRONMENT

XPLOTHOST

Can be used to specify the host and the socket for **plots( )** and **plotss( ).** It has the form *[host][:socket].*

INTERPROCESS COMMUNICATION

8124 Default socket used to talk to plotting window.

SEE ALSO

cplot(1), **plot3d**(1), **plotps**(1), **xpc**(1), **xpic**(1), **xplot**(1), **axis**(3), **crcplot**(3), **clear**(3), **curve**(3), **dline**(3), **factor**(3), **grid**(3), **hardcopy**(3), **hatch**(3), **line**(3), **mouse**(3), **number**(3), **plot**(3), **plotcrc**(3), **plots**(3), **scale**(3), **symbol**(3), **where**(3)

AUTHOR

Carl R. Crawford

BUGS

The call to **plot_space( )** is redundant because **xplot**(1) really don't support its use. However, it must be called because the display programs determine scaling parameters when a space operation is received.

NAME

plot3d – generate line drawing of three-dimensional surface

SYNOPSIS

#include <crcplot.h>

void plot3d(x,y,z,nx,ny,xmin,xmax,ymin,ymax,zmin,zmax,xlen,ylen,zlen,theta,
phi,res,pxaxis,pyaxis,pzaxis,xdir,ydir,tl,bl,xl,yl,zl)
float *x,*y,*z;
double phi,theta,res,xlen,ylen,zlen,xmin,xmax,ymin,ymax,zmin,zmax
int nx,ny,pxaxis,pyaxis,pzaxis,xdir,ydir
char *bl,*tl,*xl,*yl,*zl

DESCRIPTION

plot3d( ) is used to generate a line drawing of a specified surface at a given orientation. Hidden line removal is incorporated. Optionally, axes can be drawn along the sides of the surface. Also optionally, labels can be placed above and below the surface.

The surface is defined by the function *z(x,y)*. Vectors corresponding to *x* and *y* are required along with an array corresponding to *z*. Minimum and maximum bounds are required for the two vectors and the array. The data cannot be outside these bounds. However, the bounds can be wider than the actual data.

The viewing orientation is obtained after a first rotation about the z-axis and then using a second rotation about the new x-axis. There are no limitations on rotation angles. Some angles, particularly those that are multiple of ninety degrees, generate some strange looking results. The surface is drawn with lines parallel to the *x* and/or *y* axes.

The axis lengths are required even if the axes are not drawn. They specify a bounding box, in plot units, in which the the surface is contained. The program rotates the box using the two rotation angles and projects it mathematically onto the plotting surface. The projection is circumscribed with a rectangle whose sides are parallel to the physical *x* and *y* axes. The lower left corner is placed at the plotting origin. It is the user's responsibility to set appropriate axes lengths, origins and scale factors so that the surface is plotted in the available space. Room below and to the left of the surface has to reserved for the axes and the bottom label, if they are drawn. The top label is drawn just above the surface itself, not above the projection of the bounding box. Room has to be allowed for the top if it is used. As an example, for a scale factor of one, it has been found that the plot origin should be set to (0.75,1.0) with a call to **plot**(3). The lengths of the x-, y-, and z-axes that guarantee surfaces in the plotting region of size 10 by 10 are six, six and four, respectively.

A number of error messages are generated on standard output. The specific messages are listed below. In almost all cases, the errors are due to improper specification of the arguments. The function will exit with an error code of one in case of an error.

Hidden line removal is accomplished via an algorithm that keeps track of the upper and lower horizons along a number of discrete samples along the x-axis. Plotting time is approximately correlated with the number of steps. The number of steps should be on the order of the resolution of the output device. When the resolution is too low, the intersections between line segments can be slightly off and either gaps or crosses are generated. Therefore, far less samples are required for generating perfect plots on a monitor than on a laser printer. The *res* parameter is used to determine the number of samples. It has been found that a value of one provides a good compromise between plotting time on monitors and image quality on laser printers.

ARGUMENTS

x is an array containing the x locations of the surface. The vector must be monotonically increasing and bounded by *xmin* from below and *xmax* from above.

y is an array containing the y locations of the surface. The vector must be monotonically increasing and bounded by *ymin* from below and *ymax* from above.

z is an array of the locations of the surface to be displayed. The size of the array is *nx* by *ny*. The x-index varies the fastest. The array must be bounded by *zmin* from below and *zmax* from above.

nx is the size of the *x* vector.

ny is the size of the *y* vector.

xmin bounds the *x* vector from below.

xmax bounds the *x* vector from above.

ymin bounds the *y* vector from below.

ymax bounds the *y* vector from above.

zmin bounds the *z* array from below.

zmax bounds the *z* array from above.

xlen,ylen,zlen
are the target lengths of the three axes. The actual lengths depend on the viewing orientation and the prevailing scale factor. An axis will not be drawn if the actual length is less than one plot unit.

theta is the first rotation about the *z* axis. The units on the angle are degrees.

phi is the second rotation about the *x* axis. The units on the angle are degrees.

res determines the resolution for the hidden line removal algorithm. The actual number of steps is *res* times 2048. The product should be on the order of the number of samples in the horizontal axis of the plotting device. There is no reasonable upper limit on the parameter. The lower limit is about 0.062.

pxaxis,pyaxis,pzaxis
A non-zero value indicates that the specified axis should be plotted.

xdir,ydir A non-zero value indicates that lines parallel to the specified axis should be drawn.

tl is a text string to be plotted on top (above) of the surface. The text is centered just above the actual top of the surface.

bl is a text string to be plotted below the surface. The text is centered.

xl,yl,zl are text strings to be plotted next to the specified axes.

SEE ALSO

cplot(1), **plot3d**(1), **plotps**(1), **xpc**(1), **xpic**(1), **xplot**(1), **axis**(3), **crcplot**(3), **clear**(3), **curve**(3), **dline**(3), **factor**(3), **grid**(3), **hardcopy**(3), **hatch**(3), **line**(3), **mouse**(3), **number**(3), **plot**(3), **plotcrc**(3), **plot3d**(3), **plots**(3), **scale**(3), **symbol**(3), **where**(3)

AUTHOR

Carl R. Crawford based on code provided by Malcolm Slaney and Mani Azimi.

DIAGNOSTICS

plot3d: nx and ny must be greater than one
The *x* and *y* vectors must have at least two points in each.

plot3d: [xyz] data out of range
The data in the vectors or the arrays is not bounded by their *min* and *max* values.

plot3d: no [xyz] range
The *min* and *max* values are equal for a vector or the array.

plot3d: improper axis length
One or more of the axes lengths is less than or equal to zero.

plot3d: not enough internal resolution
: The hidden line removal algorithm needs at least 128 points. Therefore, *res* must be greater than about 0.062.

plot3d: malloc failure
: The function cannot allocate temporary space for working vectors.

plot3d: exceeded hiding boundary

plot3d: x/y mismatch
: These are internal errors and should be reported to the author.

NAME

plots, plotss, plots_fd, plots_host, plots_host – initialize plotting package

SYNOPSIS

```
#include <crcplot.h>
#include <stdio.h>

void plots()

int plotss()

void plots_host(host)
char *host;

int plotss_host(host)
char *host;

void plots_fd(fd)
FILE *fd;
```

DESCRIPTION

These functions are used to make a connection to a plotting window, allocate buffers and initialize the plotting devices before plotting can begin. One of them routines must be called before any other plotting subroutines are called.

A plotting window is a program that accepts plotting commands via a socket connection. The program that is designated for use with this package is **xplot**(1). This program, or an alternative, must be running before one of the functions listed above is called.

By default the plotting window running on the local host is called. The default socket number used for interprocess communication is listed below in the section INTERPROCESS COMMUNICATION. One might want to override the host to provide a poor-person's X windowing system. One might want to override the socket to run multiple copies of the plotting window on the same host. There are two methods to override the default host and socket number - using the environment and using arguments. In both cases the host and the socket are specified in a string of the form *[host][:socket]* where *host is the name of the host and socket* is the socket number. Either or both of the parts can be missing. Note that the colon (:) is attached to the socket number. If the host is missing, then the local host is used. If the socket is missing, then then default socket listed below is used. In the case of the environment, **XPLOTHOST** is examined for the name of the host and the port number. If host is set to the string *stdout* then the output is sent to *stdout* instead via the socket to the plotting window.

plots( ) and **plotss( )** make their connections after examining the specified environment variables. **plots( )** detects if an error has occurred during the connection prints an error message and exits. **plotss( )** returns a zero for a proper connection and one in the case of an error.

plots_host( ) and **plotss_host( )** make their connections *without* examining the specified environment variables. **plots_host( )** detects if an error has occurred during the connection, prints an error message and exits. **plotss_host( )** returns a zero for a proper connection and one in the case of an error.

plots_fd( ) will cause the **plot**(5) format output to be send to the file descriptor *fd.*

ENVIRONMENT

XPLOTHOST

Can be used to specify the host and the socket for **plots( )** and **plotss( ).** It has the form *[host][:socket].*

INTERPROCESS COMMUNICATION

8124 Default socket used to talk to plotting window.

SEE ALSO

cplot(1), **plot3d**(1), **plotps**(1), **xpc**(1), **xpic**(1), **xplot**(1), **axis**(3), **crcplot**(3), **clear**(3), **curve**(3), **dline**(3), **factor**(3), **grid**(3), **hardcopy**(3), **hatch**(3), **line**(3), **mouse**(3), **number**(3), **plot**(3), **plotcrc**(3), **plots**(3), **scale**(3), **symbol**(3), **where**(3)

AUTHOR

Carl R. Crawford

NAME

scale – compute scale factors for axis and line drawing

SYNOPSIS

#include <crcplot.h>

void scale(array,axlen,npts,inc)
float *array;
double axlen;
int npts,inc;

DESCRIPTION

scale( ) may be used to compute scale factors for processing unscaled data with subroutines **line**(3) and **axis**(3). The starting value, *fval,* representing the starting unscaled data value (either unscaled minimum or maximum), and the scaling factor, *dv,* representing the number of unscaled data units per *unit,* are returned to the next two elements (as modified by *inc*) beyond the starting address of *array* plus *npts.*

ARGUMENTS

array is the array of unscaled data to be examined.

axlen is the axis length to which the unscaled data in *array* is to be scaled. *axlen* must be greater than 1.0 *unit.*

npts defines the number of data points to be scaled in *array.* This value should not include values skipped by *inc* or the last two elements which will be set to the scaling factors. Note that if *inc* is not one, the dimension of *array* must be at least |*inc*|*(*npts*+2).

inc is the increment (absolute) used in gathering data from the unscaled *array* (for example: *inc*=1 will index each data element, *inc*=2 will index every other element, *inc*=3 will index every third data element, and so forth).

+*inc* When *inc* is positive, the first unscaled value (*fval*) approximates a minimum with a positive scale factor (*dv*).

-*inc* When *inc* is negative, the first unscaled value (*fval*) approximates a maximum with a negative scale factor (*dv*).

SEE ALSO

cplot(1), **plot3d**(1), **plotps**(1), **xpc**(1), **xpic**(1), **xplot**(1), **axis**(3), **crcplot**(3), **clear**(3), **curve**(3), **dline**(3), **factor**(3), **grid**(3), **hardcopy**(3), **hatch**(3), **line**(3), **number**(3), **plot**(3), **plotcrc**(3), **plots**(3), **scale**(3), **symbol**(3), **where**(3)

AUTHOR

Carl R. Crawford

NAME

symbol – plot alphanumeric annotations

SYNOPSIS

#include <crcplot.h>

void symbol(x,y,height,itext,angle,nc)
double x,y,height,angle;
int nc;
char *itext;

DESCRIPTION

symbol( ) may be used to plot alphanumeric annotations at various angles and sizes. Special characters not included in the standard character set of the computer may also be produced. A sample plot and chart of symbols is reproduced in the attachments.

ARGUMENTS

x,y are the starting coordinates for the lower left-hand corner of the first character to be produced.

height defines the character height in *units.*

itext For *+nc, itext* contains the alphanumeric text string to be plotted. The characters must be contiguous in *itext.* For *nc=0, itext* contains a single alphanumeric character to be plotted. For *-nc, itext* defines the integer symbol number to be plotted. If *itext* is one of the special centered symbols (0-l3) it is plotted with the *x,y* reference point as the geometric center of the symbol.

angle is the angle, in degrees that the text is rotated about the reference point before it is plotted, measured counterclockwise from the horizontal (0. is horizontal, 90. is vertical).

nc *+nc* defines the number of characters to be plotted from the text string *itext.* If *nc* is set to 999, then the length of the string will be calculated. This feature only works if the string is terminated with a zero.

- *nc*=0 indicates that *itext* contains a single character to be plotted.
- *-nc* indicates that *itext* is an integer symbol number.
- *nc*=-l specifies that the "pen" is to be up during the initial move from the present pen position to *(x,y).*
- *nc*<-1 specifies that the "pen" is to be down during the initial move from the present pen position to *(x,y).*

NOTES

The *x* and *y* parameters may contain the special value 999.0. In this case the annotation is continued at the next character position available after the last call to **symbol( ).** Note that both the *number* and *line* subroutines make internal calls to **symbol( )** and may leave the next character position at an unexpected location.

The 999.0 value may be applied to either *x* or *y* parameters separately. The user should note, however, that this may generate unexpected results, especially if a new *angle* is specified.

Character sizes which are a multiple of 7 *nibs* (7/l00", 7/l60", or 7/200" depending on the nib density of the plotter in use) produce optimum character definition at 0., 90., l80., and 270. degrees. If the *height* is specified as zero or negative, the *height* and *angle* from the last call to **symbol( )** are used.

The nominal width of characters, including spacing (from lower left-hand corner to lower left-hand corner) is the same as the height.

The character/symbol set contains five "symbols" which are special control characters rather than plotted symbols. These codes may be used in a separate call or imbedded in character strings.

backspace (BS, hex 11) causes the next available character position to be moved back one space. This can be used to produce underlined or overscored annotation.

carriage return
(CR, hex l5) causes a carriage return to be simulated. The next available character position is moved so that it is under the first character of the last line plotted. Calls specifying the *x* and *y* special 999.0 value do not affect the location to which the carriage return moves. The last call to **symbol( )** with non 999.0 coordinates determines the carriage return position. The spacing between lines is 5/7 the height of the last line plotted.

null (NUL, hex l9) is a null character and causes no space.

superscript
subscript (SUP, hex 8D)/(SUB, hex 8E) The **symbol( )** routine can operate in a subscript or superscript state. In subscript state, the subscripted character is plotted at .7 normal height and down from the middle of a normal character. In superscript state, the superscripted character is plotted at .7 normal height and up from the middle of a normal character.

Three calls to **symbol( )** can be used to plot a superscripted or subscripted text string. The first call, to enter either the subscript or superscript state, is an integer symbol mode (-*nc*) call with *itext*=111 or 110 respectively. The *height* specified in this call should be that of the main text, since the subscript/superscript text height is found by multiplying the specified value by 0.7. The actual plotting of the superscripted or subscripted text is done by a second call to **symbol( ).** This is the same as any normal call, except a *height* of zero must be specified. As long as a zero *height* is given, **symbol( )** remains in the current state. To leave the subscript or superscript state, either call **symbol( )** with a non-zero *height,* or call **symbol( )** to enter the state opposite to the one currently being used.

To plot a subscripted or superscripted text string in a single call, the appropriate control codes must be imbedded in the string. When this is done, the *height* specified should be that of the normal text. For example, to plot the formula for water, H_2O, the call to **symbol( )** should have *nc*=5, *height* equal to the desired height of the H and O, and *itext* should contain five characters, in order, H, *sub,* 2, *sup,* and 0.

SEE ALSO

cplot(1), **plot3d**(1), **plotps**(1), **xpc**(1), **xpic**(1), **xplot**(1), **axis**(3), **crcplot**(3), **clear**(3), **curve**(3), **dline**(3), **factor**(3), **grid**(3), **hardcopy**(3), **hatch**(3), **line**(3), **mouse**(3), **number**(3), **plot**(3), **plotcrc**(3), **plots**(3), **scale**(3), **symbol**(3), **where**(3)

AUTHOR

Carl R. Crawford

NAME

where – returns present pen position and scale factor

SYNOPSIS

#include <crcplot.h>

void where(xnow,ynow,dfact)
float *xnow,*ynow,*dfact;

DESCRIPTION

where( ) returns *xnow,* and *ynow,* the present "pen" position (as referenced from the current software origin) and *dfact* the current drawing factor ratio.

ARGUMENTS

xnow,ynow
: variables to be filled with the coordinates of the present "pen" position.

dfact
: variable to be set to the current drawing factor ratio.

SEE ALSO

cplot(1), **plot3d**(1), **plotps**(1), **xpc**(1), **xpic**(1), **xplot**(1), **axis**(3), **crcplot**(3), **clear**(3), **curve**(3), **dline**(3), **factor**(3), **grid**(3), **hardcopy**(3), **hatch**(3), **line**(3), **mouse**(3), **number**(3), **plot**(3), **plotcrc**(3), **plots**(3), **scale**(3), **symbol**(3), **where**(3)

AUTHOR

Carl R. Crawford

NAME

example – example programs using the CRC Plotting Package

DESCRIPTION

The following programs demonstrate the capabilities of the CRC Plotting Package and programs that themselves utilize the package.

ex1.c draws a simple house using **plot**(3).

ex2.c draws two versions of house while demonstrating **where**(3) and **factor**(3).

ex3.c demonstrates the use of **symbol**(3).

ex4.c demonstrates the use of **number**(3).

ex5.c demonstrates the use of **scale**(3).

ex6.c demonstrates the use of **line**(3).

ex7.c demonstrates the use of **axis**(3).

ex8.c demonstrates the use of **curve**(3).

ex9.c shows a complex example using **line**(3), **axis**(3) and **scale**(3).

ex10.c shows another complex plotting example.

ex11.c demonstrates the use of **grid**(3).

ex12.c demonstrates the use of **qplot**(1).

ex13.c demonstrates the use of **cplot**(1).

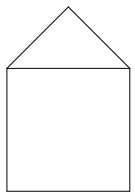
ex14.c demonstrates the use of **plot3d**(1).

SEE ALSO

cplot(1), **plot3d**(1), **plotps**(1), **qplot**(1), **sunpic**(1), **sunplot**(1), **xpc**(1), **xpic**(1), **xplot**(1), **axis**(3), **crcplot**(3), **clear**(3), **curve**(3), **dline**(3), **factor**(3), **grid**(3), **hardcopy**(3), **hatch**(3), **line**(3), **number**(3), **plot**(3), **plotcrc**(3), **plots**(3), **scale**(3), **symbol**(3), **where**(3)

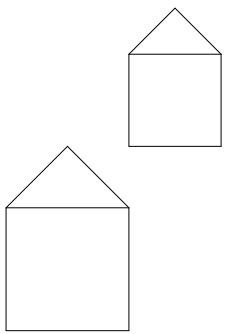
AUTHOR

Carl R. Crawford



```
/*
        ex1.c - plot example - house
*/
#ident "$Id: ex1.c,v 1.4 2007/05/22 15:33:04 ccrawford Exp $"
#include <crcplot.h>
#include <stdlib.h>

int
main()
{
/*
        initialize plotting
*/
        plots();
        clear();
/*
        move to origin from any previous position, pen up
*/
        plot(2.,2.,-3);
/*
        draw square
*/
        plot(1.,0.,2);
        plot(1.,1.,2);
        plot(0.,1.,2);
        plot(0.,0.,2);
/*
        draw roof
*/
        plot(0.,1.,3);
        plot(.5,1.5,2);
        plot(1.,1.,2);
/*
        end of plot and plotting
*/
        plot(0.,0.,999);
        exit(0);
}
```

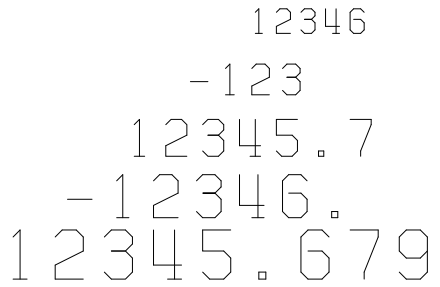
```
values returned by where:   0.00   1.00   1.00
```




```
/*
        ex2.c -factor and where example
*/
#ident "$Id: ex2.c,v 1.4 2007/05/22 15:33:04 ccrawford Exp $"
#include <stdio.h>
#include <crcplot.h>
#include <stdlib.h>

static void
house()
{
/*
        origin and scale determined externally draw house
*/
        plot(0.,0.,3);
        plot(1.,0.,2);
        plot(1.,1.,2);
        plot(0.,1.,2);
        plot(0.,0.,2);
/*
        draw roof
*/
        plot(1.,1.,3);
        plot(.5,1.5,2);
        plot(0.,1.,2);
}

int
main()
{
        float xnow,ynow,dfact;
/*
        initialize plotting
*/
        plots();
        clear();
        plot(1.0,1.5,-3);
/*
        draw house
*/
        house();
/*
        get current position and factor and print them
*/
        where(&xnow, &ynow, &dfact);
        fprintf(stderr,
                "values returned by where: %6.2f %6.2f %6.2f\n",
                xnow, ynow, dfact);
/*
        draw house at new position
*/
        plot(1.0,1.5,-3);
        factor(.75);
        house();
/*
        end of plotting
*/
        plot(0.,0.,999);
        exit(0);
}
```

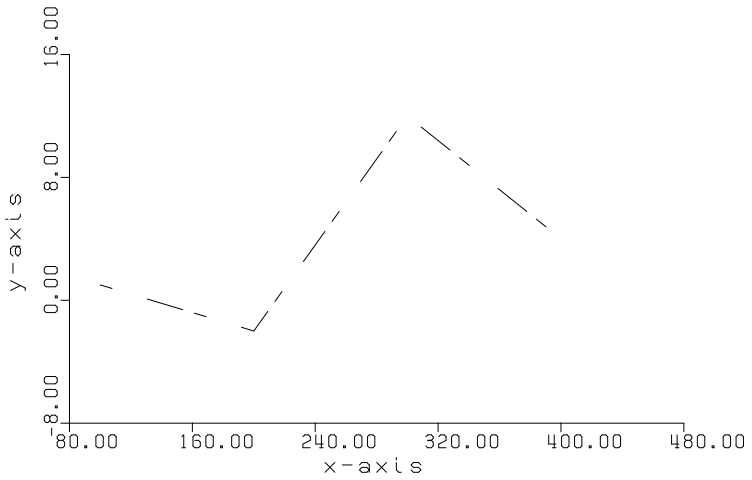
```
/*      ex3.c - symbol example
*/
#ident "$Id: ex3.c,v 1.4 2007/05/22 15:33:04 ccrawford Exp $"
#include <math.h>
#include <crcplot.h>
#include <stdlib.h>

int
main()
{
        float x,y,angle,height,rad;
        int i;

        plots();
        clear();
        angle = 0.0;
        height = 0.14;

/*      position pen to center
*/
        plot(4.,4.,-3);

/*      draw symbols
*/
        for(i=0;i<8;i++){
                rad = 0.017453*angle;
                x = 0.5*cos(rad);
                y = 0.5*sin(rad);
                symbol(x,y,height,"crcplot",angle,7);
                height = height + 0.028;
                angle = angle + 45.0;
        }

/*      end plotting
*/
        plot(0.,0.,999);
        exit(0);
}
```

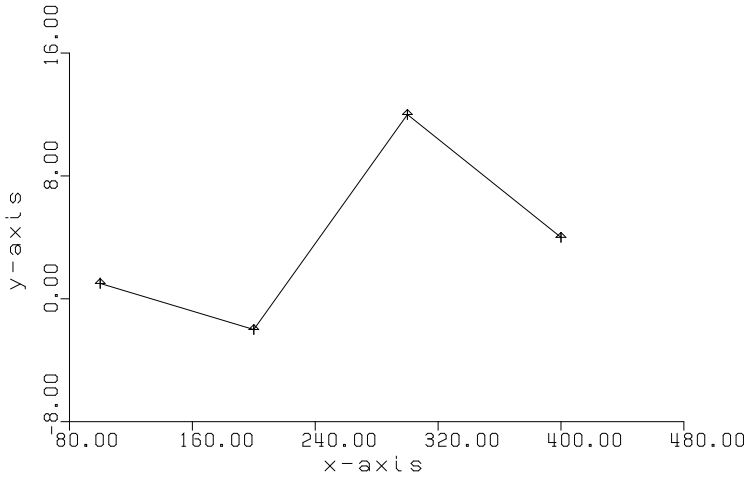
```
/*
        ex4.c - number example
*/
#ident "$Id: ex4.c,v 1.4 2007/05/22 15:33:04 ccrawford Exp $"
#include <crcplot.h>
#include <stdlib.h>

int
main()
{
        float value = 12345.6789;
/*
        initialize plotting
*/
        plots();
        clear();
        plot(1.0,1.0,-3);
/*
        plot numeric values
*/
        number(0.0,0.0,0.40, value, 0.0,3);
        number(1.5,1.5,0.25, -value, 0.0,-3);
        number(2.0,2.0,0.20, value, 0.0,-1);
        number(1.0,1.0,0.30, value, 0.0,1);
        number(0.5,0.5,0.35, -value, 0.0,0);
/*
        end of plotting
*/
        plot(0.,0.,999);
        exit(0);
}
```



```
y[0] =   1.00
y[1] =  -2.00
y[2] =  12.00
y[3] =   4.00
y[4]= -2.00,(FVAL as determined by scale)
y[5]=  2.00,(DV as determined by scale)
```

```
/*
        ex5.c - scale example
*/
#ident "$Id: ex5.c,v 1.4 2007/05/22 15:33:04 ccrawford Exp $"
#include <stdio.h>
#include <crcplot.h>
#include <stdlib.h>

int
main()
{
        static float y[] = {1.,-2.,12.,4.,0.,0.};
        int i;
/*
        initialize plotting
*/
        plots();
        clear();
/*
        scale 4 values to a 7. inch axis
*/
        scale(y,7.,4,1);
/*
        print results of scaling
*/
        for(i=0;i<4;i++)
                fprintf(stderr,"y[%d] = %6.2f\n",i,y[i]);

        fprintf(stderr,
                "y[4]=%6.2f,(FVAL as determined by scale)\n",y[4]);
        fprintf(stderr,
                "y[5]=%6.2f,(DV as determined by scale)\n",y[5]);
/*
        end of plotting
*/
        plot(0.,0.,999);
        exit(0);
}
```

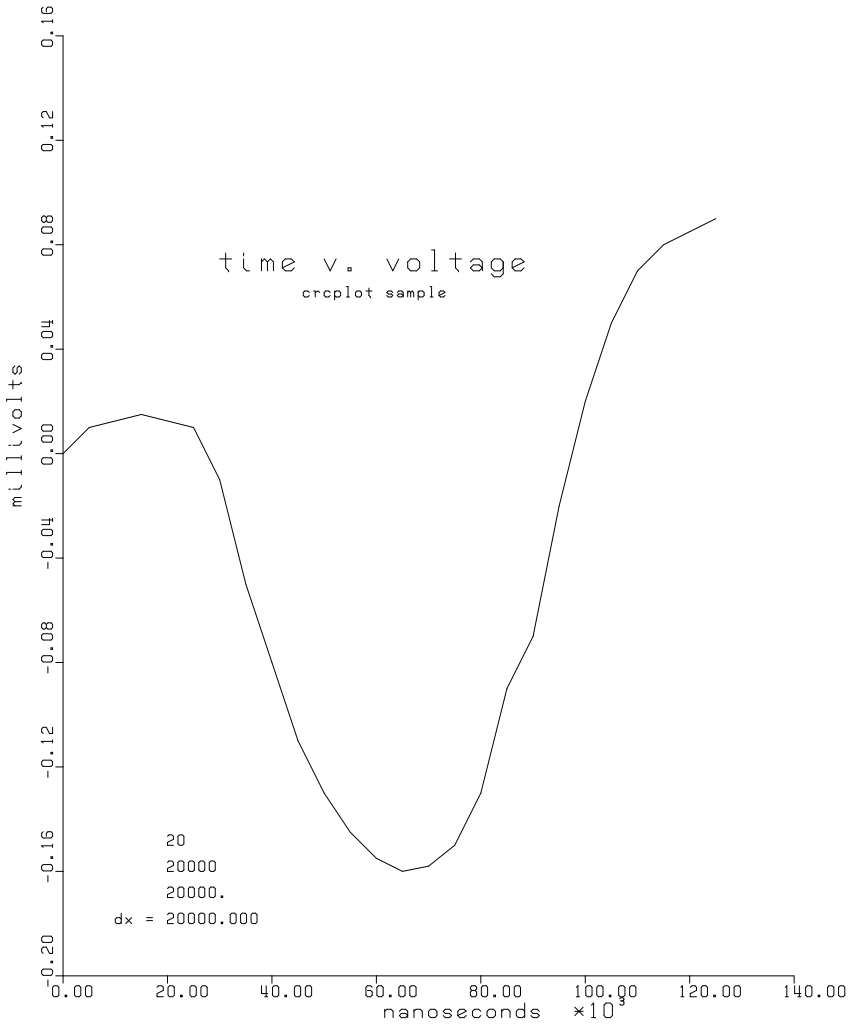




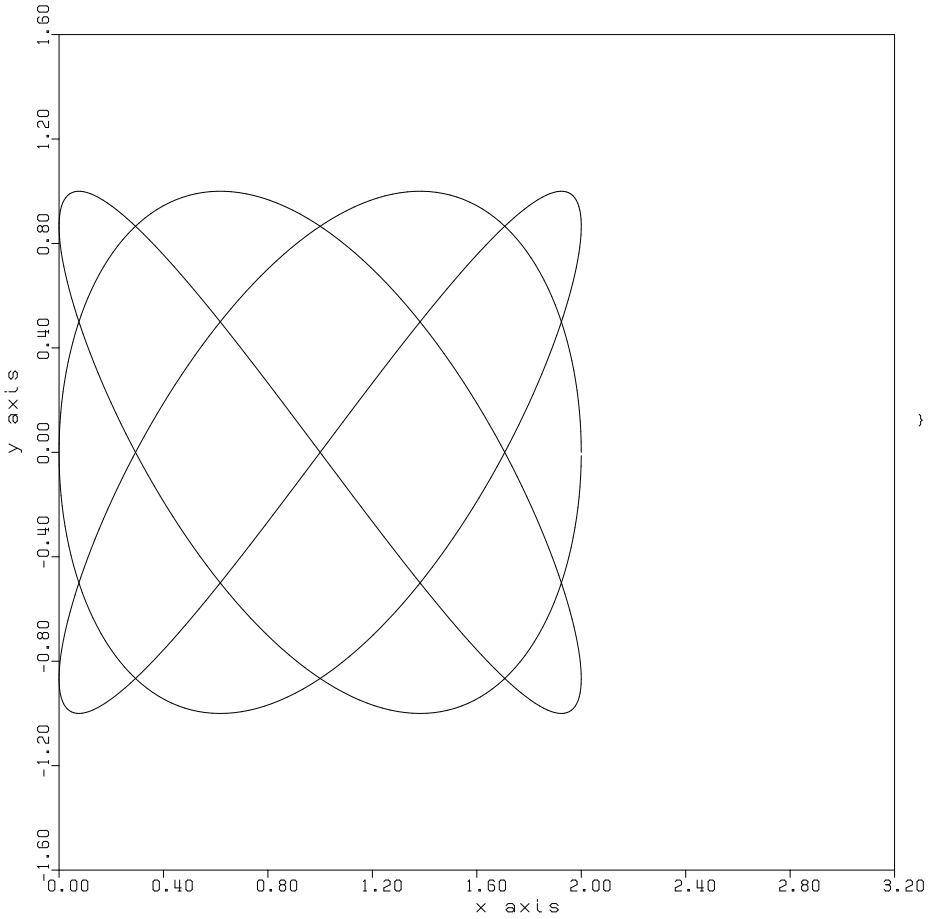

```
/*
        ex6.c - dline and line example
*/
#ident "$Id: ex6.c,v 1.4 2007/05/22 15:33:04 ccrawford Exp $"
#include <crcplot.h>
#include <stdlib.h>

int
main()
{
        static float x[] = { 100.,200.,300.,400.,0.,0.};
        static float y[] = { 1.,-2.,12.,4.,0.,0.};
        static float dsh[] = { .1, .3};
        static float gap[] = { .3, .1};
/*
        initialize plotting
*/
        plots();
        clear();
/*
        reorigin the entire plot to allow the
        axis label to be in the plottable area
*/
        plot(1.5,1.5,-3);
/*
        scale x values to a 5. inch axis and
        y values to a 4. inch axis and plot them
*/
        scale(x,5.,4,1);
        scale(y,3.,4,1);

        axis(0.,0.,"y-axis",999,3.,90.,y[4],y[5]);
        axis(0.,0.,"x-axis",-999,5., 0.,x[4],x[5]);
        line(x,y,4,1,1,6);

        plot(0.0,4.0,-3);
        axis(0.,0.,"y-axis",999,3.,90.,y[4],y[5]);
        axis(0.,0.,"x-axis",-999,5., 0.,x[4],x[5]);
        dline(x,y,4,dsh,gap,2);
/*
        end of plotting
*/
        plot(0.,0.,999);
        exit(0);
}
```

```
/*      ex7.c - axis example
*/
#ident "$Id: ex7.c,v 1.4 2007/05/22 15:33:04 ccrawford Exp $"
#include <crcplot.h>
#include <stdlib.h>

int
main()
{
        static float y[] = {1.,-2.,12.,4.,0.,0.};
    /*
        initialize plotting
    */
        plots();
        clear();
        plot(1.,1.,-3);
    /*  scale 4 values to a 7. inch axis
    */
        scale(y,7.,4,1);
    /*  draw axis
    */
        axis(0.5,0.5,"z-axis",6,7.,45.,y[4],y[5]);
    /*  end of plotting
    */
        plot(0.,0.,999);
        exit(0);
    }
```

```
/*      ex8.c - curve example
*/
#ident "$Id: ex8.c,v 1.4 2007/05/22 15:33:04 ccrawford Exp $"
#include <crcplot.h>
#include <stdlib.h>

int
main()
{
        static float x[] = {1.,2.,3.,4.,5.,6.,7.,7.3,8.};
        static float y[] = {4.,3.,2.,3.,4.,4.,4.,3.,4.};
        int i;

    /*  initialize
    */
        plots();
        clear();
        plot(1.,1.,-3);

    /*  draw dashed curve
    */
        curve(x,y,9,-.05);

    /*  draw symbol at each data point
    */
        for(i=0;i<9;i++)
                symbol(x[i],y[i],.14,(char *)0,0.,-1);

    /*  terminate plotting
    */
        plot(0.,0.,999);
        exit(0);
}
```




```
/*
        ex9.c - demonstrates line, axis and scale
*/
#ident "$Id: ex9.c,v 1.4 2007/05/22 15:33:04 ccrawford Exp $"
#include <crcplot.h>
#include <stdlib.h>

int
main()
{
        int i;

        static float x[] = {
                0.,5.,15.,25.,30.,35.,40.,45.,50.,55.,60.,65.,
                70.,75.,80.,85.,90.,95.,100.,105.,110.,115.,120.,
                125.,0.,0.0};
        static float y[] = {
                0.,10.,15.,10.,-10.,-50.,-80.,-110.,-130.,-145.,
                -155.,-160.,-158.,-150.,-130.,-90.,-70.,-20.,20.,
                50.,70.,80.,85.,90.,0.0,0.0};
/*
        alter actual data scales
*/
        for(i=0;i<24;i++){
                x[i] = x[i] * 1000.;
                y[i] = y[i] / 1000.;
        }
/*
        initialize for plotting
*/
        plots();
        clear();
        plot(1.0,.75,-3);
        factor(.85);
/*
        determine scaling factors
*/
        scale(x,7.,24,1);
        scale(y,9.,24,1);
/*
        plot the data
*/
        line(x,y,24,1,0,0);
/*
        draw axes
*/
        axis(0.,0.,"nanoseconds",-11,7.,0.,x[24],x[25]);
        axis(0.,0.,"millivolts",10,9.,90.,y[24],y[25]);
/*
        annotate the plot
*/
        symbol(.5,.5,.1,"dx = ",0.,5);
        number(999.,999.,.1,x[25],0.,3);
        number(1.,.75,.1,x[25],0.,0);
        number(1.,1.,.1,x[25],0.,-1);
        number(1.,1.25,.1,x[25],0.,-4);
        symbol(2.3,6.5,.1,"crcplot sample",0.,14);
        symbol(1.5,6.75,.2,"time v. voltage",0.,15);
/*
        end of plotting
*/
        plot(0.,0.,999);
        exit(0);
}
```

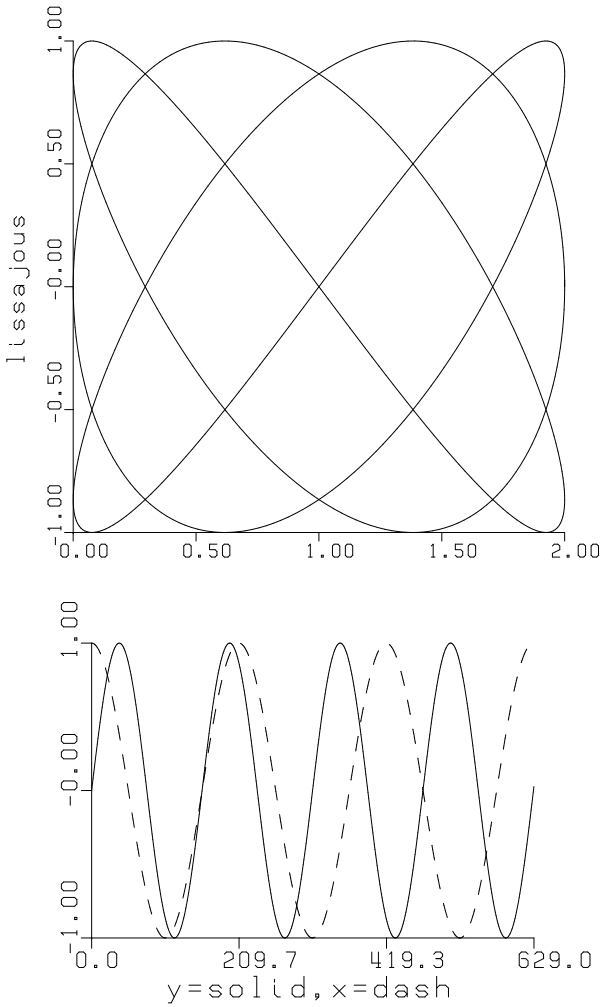



```
/*
        ex10.c - demonstrates line, axis and scale
*/
#ident "$Id: ex10.c,v 1.4 2007/05/22 15:33:04 ccrawford Exp $"
#include <math.h>
#include <crtplot.h>
#include <stdlib.h>

int
main()
{
        int i;
        float x[631],y[631];

        plots();
        clear();
        plot(1.0,1.0,-3);
        factor(.85);

        for(i=0;i<630;i++){     /* compute data */
                x[i] = cos(3.0 * i / 100) + 1.0;
                y[i] = sin(4.0 * i / 100);
        }

        scale(y,8.0,629,1);     /* get scaling information */
        scale(x,8.0,629,1);

        axis(0.,0.,"x axis",-6,8.,0.0,x[629],x[630]);
        axis(0.,0.,"y axis",6,8.,90.0,y[629],y[630]);
        plot(0.0,8.0,3);        /* draw border */
        plot(8.0,8.0,2);
        plot(8.0,0.0,2);
        symbol(2.5,8.5,.2,"lissajous figure",0.0,16);
        line(x,y,629,1,0,0);

        plot(0.0,0.0,999);      /* terminate plotting */
        exit(0);
}
```

```
/*
        ex11.c - examples using grid

*/
#ident "$Id: ex11.c,v 1.4 2007/05/22 15:33:04 ccrawford Exp $"
#include <crcplot.h>
#include <stdlib.h>

int
main()
{
        float dx=1.0,dy=2.0;
        static float dxa[] = {.1,.2,.3,.4};

        plots();
        clear();
        grid(1.0,1.0,4,&dx,3,&dy);
        grid(6.0,1.0,1004,dxa,3,&dy);
        plot(0.0,0.0,999);
        exit(0);
}
```

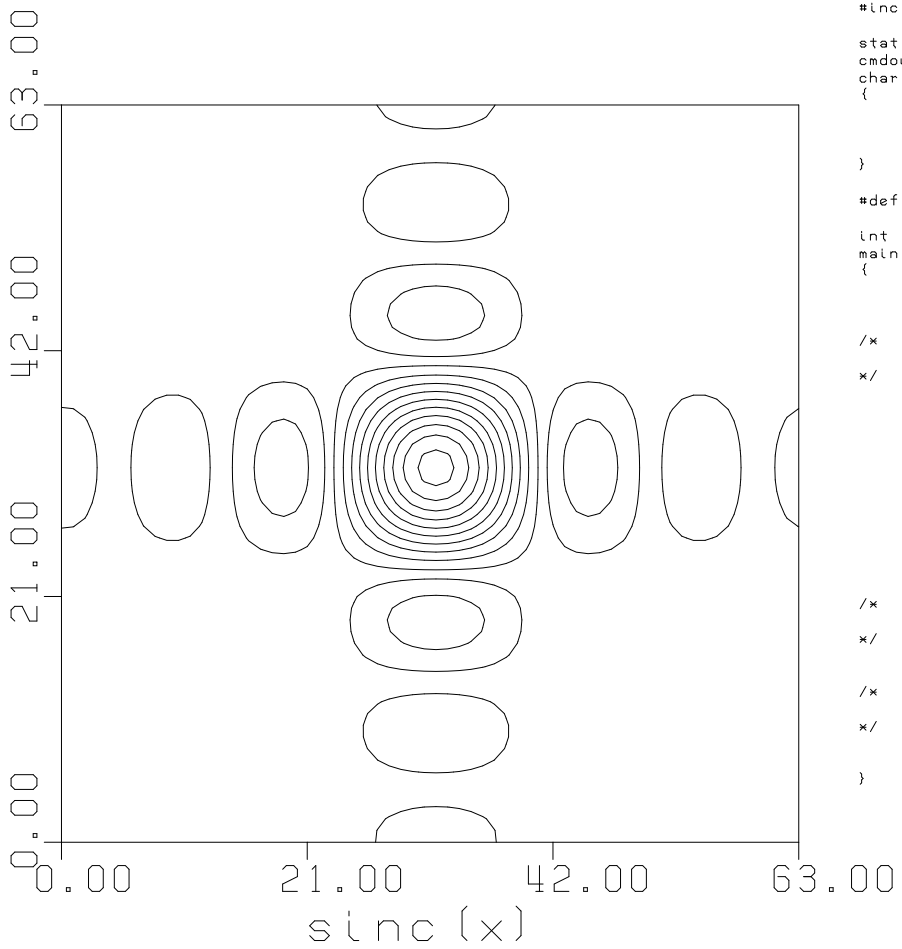


```
../plot/qplot -@ y ylen=2 xl=y=solid,x=dash yp=-.5 scale=.6 xp=1
../plot/qplot -@ x ylen=2 -ead yp=-.5 scale=.6 xp=1
../plot/qplot -@ y=y x=x yl=lissajous len=4 yp=3 -e scale=.5 xp=1
```



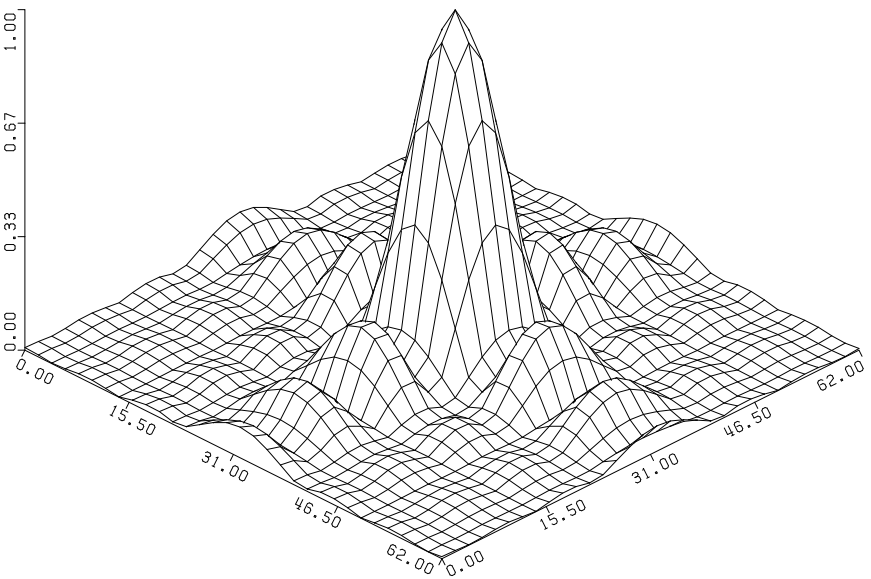

```
/*
        ex12.c - demonstrates qplot
*/
#ident "$Id: ex12.c,v 1.3 2007/05/22 15:33:04 ccrawford Exp $"
#include <math.h>
#include <crcplot.h>
#include <unistd.h>
#include <stdlib.h>
#include <parse.h>

static void
cmdout(cmd)
char *cmd;
{
        fputs(cmd,stderr);
        fputs("\n",stderr);
        system(cmd);
}

int
main(){
        int i;
        float x[631],y[631];
        char *cmd;
/*
        generate data and save in external files
*/
        for(i=0;i<630;i++){     /* compute data */
                x[i] = cos(3.0 * i / 100) + 1.0;
                y[i] = sin(4.0 * i / 100);
        }
        parse_wda(x,630,1,"x");
        parse_wda(y,630,1,"y");
/*
        simulation prompt invocation of qplot
*/
        cmd = "../plot/qplot -@ y ylen=2 xl=y=solid,x=dash yp=-.5 scale=.6 xp=1";
        cmdout(cmd);
        cmd = "../plot/qplot -@ x ylen=2 -ead yp=-.5 scale=.6 xp=1";
        cmdout(cmd);
        cmd = "../plot/qplot -@ y=y x=x yl=lissajous len=4 yp=3 -e scale=.5 xp=1";
        cmdout(cmd);
/*
        clean up
*/
        unlink("x.da");
        unlink("y.da");
        exit(0);
}
```

```
../plot/cplot -@ z=sinc,d -f xl=sinc\(x\)
```




```
/*
        ex13.c - demonstrates cplot
*/
#ident "$Id: ex13.c,v 1.3 2007/05/22 15:33:04 ccrawford Exp $"
#include <math.h>
#include <crcplot.h>
#include <unistd.h>
#include <stdlib.h>
#include <parse.h>

static void
cmdout(cmd)
char *cmd;
{
        fputs(cmd,stderr);
        fputs("\n",stderr);
        system(cmd);
}

#define N 64

int
main()
{
        int i,j,k;
        float sx,sy,x,y,z[N*N];
        char *cmd;
/*
        generate data and save in external file
*/
        for(k=i=0;i<N;i++) {
                y = (i - N/2.) / N * M_PI * 7.0;
                if(i == N/2) sy = 1.0;
                else sy = fabs(sin(y)/y);
                for(j=0;j<N;j++) {
                        x = (j - N/2.) / N * M_PI * 7.0;
                        if(j == N/2) sx = 1.0;
                        else sx = fabs(sin(x)/x);
                        z[k++] = sx * sy;
                }
        }
        parse_wda(z,N,N,"sinc");
/*
        simulation prompt invocation of cplot
*/
        cmd = "../plot/cplot -@ z=sinc,d -f xl=sinc\\(x\\)";
        cmdout(cmd);
/*
        clean up
*/
        unlink("sinc.da");
        exit(0);
}
```

```
../plot/plot3d -© z=sinc,d zl=sinc\(x\) dir=xy skip=1 scale=.8
```




```
/*
        ex14.c - demonstrates plot3d
*/
#ident "$Id: ex14.c,v 1.3 2007/05/22 15:33:04 ccrawford Exp $"
#include <math.h>
#include <unistd.h>
#include <stdlib.h>
#include <crcplot.h>
#include "parse.h"

#define N 64

static void
cmdout(cmd)
char *cmd;
{
        fputs(cmd,stderr);
        fputs("\n",stderr);
        system(cmd);
}

int
main()
{
        int i,j,k;
        float sx,sy,x,y,z[N*N];
        char *cmd;
/*
        generate data and save in external file
*/
        for(k=i=0;i<N;i++) {
                y = (i - N/2.) / N * M_PI * 7.0;
                if(i == N/2) sy = 1.0;
                else sy = fabs(sin(y)/y);
                for(j=0;j<N;j++) {
                        x = (j - N/2.) / N * M_PI * 7.0;
                        if(j == N/2) sx = 1.0;
                        else sx = fabs(sin(x)/x);
                        z[k++] = sx * sy;
                }
        }
        parse_wda(z,N,N,"sinc");
/*
        simulation prompt invocation of plot3d
*/
        cmd = "../plot/plot3d -© z=sinc,d zl=sinc\\(x\\) dir=xy skip=1 scale=.8";
        cmdout(cmd);
/*
        clean up
*/
        unlink("sinc.da");
        exit(0);
}
```

Symbol Routine Character Definitions

0	□	26		52	4	78	N	104	h	130	π
1	◷	27	∫	53	5	79	O	105	i	131	Φ
2	◬	28	⊃	54	6	80	P	106	j	132	Θ
3	+	29	∨	55	7	81	Q	107	k	133	Ψ
4	×	30	~	56	8	82	R	108	l	134	×
5	◇	31	≈	57	9	83	S	109	m	135	ω
6	✦	32		58	:	84	T	110	n	136	λ
7	⋈	33	¡	59	;	85	U	111	o	137	α
8	Z	34	¨	60	<	86	V	112	p	138	δ
9	⋎	35	#	61	=	87	W	113	q	139	∈
10	¤	36	$	62	>	88	X	114	r	140	η
11	✳	37	%	63	?	89	Y	115	s	141	SUP
12	⧖	38	&	64	@	90	Z	116	t	142	SUB
13	⏐	39	'	65	A	91	[	117	u	143	∑
14	✡	40	(	66	B	92	\	118	v	144	÷
15	_	41	)	67	C	93	]	119	w	145	≤
16	⏐	42	*	68	D	94	^	120	x	146	≥
17	BS	43	+	69	E	95	_	121	y	147	△
18	}	44	,	70	F	96	∞	122	z	148	¢
19	≡	45	-	71	G	97	a	123	{	149	—
20	→	46	.	72	H	98	b	124	←	150	⏐
21	CR	47	/	73	I	99	c	125	}	151	⍑
22	≠	48	0	74	J	100	d	126	↑	152	⌜
23	±	49	1	75	K	101	e	127	↓	153	‡
24	{	50	2	76	L	102	f	128	¬		
25	NUL	51	3	77	M	103	g	129	μ		

14 September 2011