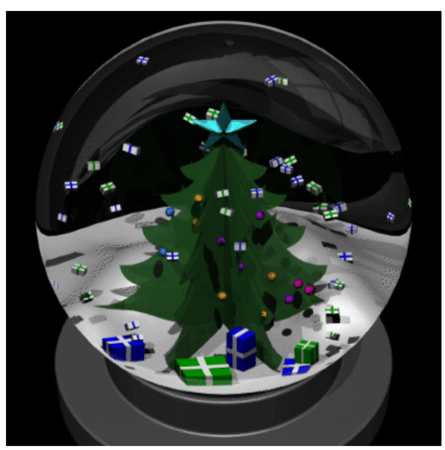
CCS-ECL Libraries

Reference Manual - I

for

Kernel and Interface Libraries



About This Manual

There are five program levels in CCS-ECL libraries, they are:

- interface (Top level and a single statement for all)
- user (easy to use)
- assemble (assembled from kernel and can be further assembled to user level)
- fuzzy (not for user. soft tissue between interface and kernel)
- kernel (more efficient)

These levels are further divided into many categories, such as colors, error, message, image, input/output, memory, network, etc.

Functions and subroutines at interface and user level are the most simple routine calls. Most of them can achieve a complex goal by using only one statement, such as (*img->header_handle)(), (*img->std_swif)() in CCS library, and get_pic(), LoadGXimage() in panel library. See program guide for detailed information.

Most interface functions require format_init() to initial the U_IMAGE structure if the U_IMAGE structure is invoked in that function call.

The U_IMAGE, Image, and image_information share the same basic structure. The U_IMAGE contains the basic image structure (including X window base), and other two contain X color handling structures. Also, the X window base in these image structure is same as all the panel and window structure, such as AnyWindow, Panel, AnyParts, Button, etc.

If ** symbol appear on any function or sub-routine name line after "—", this function or sub-routine is not encouraged to be directly used in user's programming. Most of them are fuzzy layers which are the interface between kernel and user interface. Try to find corresponding interface.

Any function or sub-routine that has a * symbol on its name line after "—" is suggested by using interface call instead of using them directly.

Some functions and sub-routines are not released to public. These functions and sub-routines are available to all HIPS users and LBL users. To be able to use them, contact the responsible people shown in the copyright page in source codes.

Name:

CalcSubWinMean — calculate mean value in a given area of RLE (interline 24-bit) images.

Synopsis:

```
CalcSubWinMean(    byte    *rle_in,
                   byte    *rle_out,
                   int     rle_line_width,
                   int     x0, y0,
                   int     sub_win_w, sub_win_h)
```

Arguments:

rle_in	source data pointer
rle_out	destination data pointer
rle_line_width	RLE image width
x0 and y0	origin of sub-window

When x0 & y0 zeros, rle_in and rle_out point to beginning of sub buffer.
Otherwise, rle_in and rle_out should point to beginning of original array.

sub_win_w	
sub_win_h	sub-window dimensions

Description:

The input must be in RLE image format.

It returns mean value in a given area to rle_out. For example: input contains 50% red and 50% green components, the result in rle_out is the yellow component

Program Level:

kernel

Related Functions:

Name:

CloseColor_in_Map — find a closest color in colormap and return the index.

Synopsis:

```
CloseColor_in_Map(    cmap_t*    colormap[],
                      int        colormap_len,
                      int        vr, vg, vb,
                      int        error         )
```

Arguments:

colormap	user's color map
colormap_lem	user's color map length
vr, vg, vb	color value in red, green, and blue for matching
error	maximum difference. 0 means to match exactly.

Description:

CloseColor_in_Map compares vr, vg, and vb values with colormap[0][index], colormap[1][index], and colormap[2][index] values according error value. When error is zero, the vr, vg, and vb have to be exactly equal to colormap[0][index], colormap[1][index], and colormap[2][index], respectively. If error is greater than 0, then CloseColor_in_Map() function will try to find the closest color set in the color map in the error range and return an index. If the comparison difference is greater than error, it returns -1 to indicate failure.

Error function is:

(abs(vr - cr[i]) + abs(vg - cg[i]) + abs(vb - cb[i]))

cr = colormap[0], cg = colormap[1], and cb = colormap[2]

The default error value is 384 (average 50% error for each component)

Program Level:

kernel

Related Functions:

Name:

DBFourier — Fast Fourier Transform in double floating format
DBvfft2d
DBvrft2d
DBvfft3d
DBvrft3d — double floating VFFT in 2D and 3D domain

Synopsis:

```
DBFourier(   DBCOMPLEX*          in_buf,
             unsigned            number_of_data_points,
             DBCOMPLEX*          out_buf          )


DBvfft2d(    VType*     src_buf,
             unsigned   x, y,
             VType*     dst_buf         )

DBvrft2d(    VType*     src_buf,
             unsigned   x, y,
             VType*     dst_buf         )

DBvfft3d(    double*             src_buf,
             int                 x, y, z,
             DBCOMPLEX*          dst_buf          )

DBvrft3d(    DBCOMPLEX*          src_buf,
             int                 x, y, z,
             double*             dst_buf          )
```

Arguments:

src_buf	sources buffer (data)
x, y, z	number_of_data_points in each dimension
dst_buf	destination buffer (result)

Description:

Program Level:

kernel and assemble

Related Functions:

Fourier(), vfft

Name:

Fourier — compute one dimensional Fourier Transform

Synopsis:

```
Fourier(    COMPLEX*   in_buf,
            unsigned   number_of_data_points,
            COMPLEX*   out_buf         )
```

Arguments:

Description:

Program Level:

assemble

Related Functions:

vfft

Name:

GetItem — get single variable from a string buffer

Synopsis:

```
convts*
GetItem(    char          option,
            char*         in_buf          )
```

Arguments:

option	converting type, such as: "b", "c", "d", "f", ...
in_buf	source string buffer contains input strings.

Description:

Program Level:

user

Related Functions:

Name:

LongSwap — swap the four bytes order in a long integer.

Synopsis:

long
LongSwap(long val)

Arguments:

val	the long integer value will be swapped

Description:

Program Level:

kernel

Related Functions:

ShortSwap()

Name:

QuickSort — quick sort procedure

Synopsis:

```
QuickSort(int       MinQS, MaxQS, MaxSort,
          QSCell    *array                  )
```

Arguments:

Description:

Program Level:

assemble

Related Functions:

Name:

ReadRGBMap — read RGB binary color map

Synopsis:

```
ReadRGBMap(U_IMAGE   *img,
           color_cell  *cmbuf,
           int         mlen,
           bool        OsameI      )
```

Arguments:

img	image structure pointer
cmbuf	color map buffer
mlen	color map length
OsameI	not used

Description:

ReadRGBMap() reads a binary RGBRGB... color map from input and converts it into regular color map (RRRR..., GGGG..., BBBB...)

Return:

1 for successful, or 0 in failure.

Program Level:

kernel

Related Functions:

read_hex_rgbmap(), read_ras_cmap(), rgbmap_to_othermap(), vff_def_cmap()

Name:

ResetLKT — reset lookup table

Synopsis:

```
ResetLKT( LKT        *lkt,
          U_IMAGE    *img        )
```

Arguments:

lkt	lookup table pointer
img	image structure pointer

Description:

Program Level:

assemble

Related Functions:

Name:

ShortSwap — swap two bytes in a short integer

Synopsis:

short
ShortSwap(short val)

Arguments:

Description:

Program Level:

kernel

Related Functions:

LongSwap()

Name:

ShowA — display content of a quick sort array

Synopsis:

ShowA(QSCell *A, int Size)

Arguments:

A	array in quicksort format
Size	array size

Description:

Program Level:

assemble

Related Functions:

Name:

To_8 — convert a true color image to a Pseudo color image.

Synopsis:

To_8(	U_IMAGE	*img,	
	cmap_t	*colormap[],	
	bool	quant,	
	int	num_colors	)

Arguments:

*img	image structure pointer
colormap	return a colormap for 8-bit (pseudo color) image
quant	default is do dithering if quant is False
nc	number colors you want to generate (maximum is 256)

Description:

To_8() is a conversion tool to convert true color image to pseudo color image.

Program Level:

kernel

Related Functions:

dither

Name:

System variables and some internal routines

Synopsis:

_BLUE_to_GRAY
_BUF_P
_ByteShift
_ColorCells
_DBNfactors
_DBW_factors
_DataType
_FORTRAN
_F_Offset
_GREEN_to_GRAY
_Gif89
_GifScreen
_ITypeName
_MaxShort
_MinShort
_Mversion
_Nfactors
_PBuffer
_PBuffer
_QSArray
_RED_to_GRAY
_RastSWidth(struct rasterfile*rast_hd)
_W_factors
_ZLevel
_cmpx_in
_cmpx_out
_debug
_dw_im
_dw_re
_fh_buf
_freeboxes
_io_test_msg
_msg_hdr
_oPBSize
_r_cmap
_reg_cmap
_row_dpy_mode
_sonion
_usedboxes
_work_space_init

Arguments:

N/A

Description:

A few of these variables are useful for programming; they are:

```
int             RED_to_GRAY, GREEN_to_GRAY, BLUE_to_GRAY
        color to gray-scale conversion ratio registers
char*           ITypeName[]   /*image type names          */
char*           Progname,
bool            debug         /* kernel debugging level   */
sht_cmap_t      *r_cmap       /* RLE style color map pointer*/
cmap_t          *reg_cmap[]   /* regular color map arrays */
```

Others are not suggested to use.

For information about RLE, TIFF and other library functions and subroutines, see documents in their libraries.

Program Level:

None

Related Functions:

Name:

alloc_2d_discrete — allocate 2-dimensional array line by line.

Synopsis:

```
VType*
alloc_2d_discrete(      int      cols,
                        int      rows,
                        int      elem_size    )
```

Arguments:

cols	line width
rows	number of lines
elem_size	size of buffer type, such as sizeof(int) or sizeof(char*)

Description:

alloc_2d_discrete() first allocates rows pointer buffer, then allocates each pointer by size of cols times elem_size.

Program Level:

kernel

Related Functions:

free_2d_discrete(), nzalloc(), zalloc(), verify_buffer_size(), pointer_buffer_size(), last_pointer_pos()

Name:

any_ilc_to_rle — convert any RGB, ARGB, BGR, or BGRA format to ILL (RLE) format.

Synopsis:

```
any_ilc_to_rle(     byte        *obp, *ibp,
                    U_IMAGE     *img,
                    int         channels,
                    bool        reverse       )
```

Arguments:

obp	output RLE buffer pointer
ibp	input ILC buffer pointer
channels	input ILC channels. 3 for RGB or BGR; 4 for Alpha
reverse	reverse BGR to RGB. No overhead at all

Description:

Program Level:

kernel

Related Functions:

ilc_transf()

Name:

any_to_seplane — convert different types of images to a separated plane image.

Synopsis:

```
any_to_seplane(    U_IMAGE    *img,
                   int        channels,
                   bool       reverse,
                   cmap_t     *colormap    )
```

Arguments:

img	image structure pointer
channels	incoming image channels
reverse	reverse RGB to BGR
colormap	color map 8-bit input. NULL for true color image

Description:

Program Level:

kernel

Related Functions:

Name:

arget — return a value if next string in argv array is a number (digital).

Synopsis:

```
arget(   int       argc,
         char**    argv,
         int       *row, *filed   )
```

Arguments:

argc	number of arguments in argv list
argv	argument string lists
row	current argument string is at nth (row) string in argv lists
field	nth (field) character in the current string

Description:

arget() returns the ASCII value of the first character in the next available string, if that string is a numerical string, and row and field may or may not be changed depending on current string content. Otherwise, it returns 0, row and field are not changed.

e.g.:

argc = 4
argv[0] = program name
argv[1] = -px
argv[2] = 100
argv[3] = -B
*row = 1
*field = 2
current string = argv[1] + 2 = “x”

Since “x” is not a digit, but next string in argv list is. Therefore, arget() returns 48 (‘1’), advances *row = 2, and set field = 0.

Program Level:

kernel

Related Functions:

avset(), parse_argus()

Name:

available_type — find image ID by name

Synopsis:

```
int
available_type(char*        image_name     )
```

Arguments:

image_name is any string which describes an image type, such as tiff, Tiff, TIFF, Hips, ...

Description:

available_type() returns image type ID if image name is in current image list. For example, available_type("rle") returns RLE (value is 3).

Program Level:

kernel

Related Functions:

char *ITypeName[]

Name:

avset — set next available string to satisfy string or number.

Synopsis:

```
avset(   int        argc,
         char**     argv,
         int        *nth_argv, *nth_filed,
         bool       not_number    )
```

Arguments:

argc	number of arguments in argv list
argv	argument string lists
nth_argv	current argument string is at nth string in argv lists
nth_field	nth character (field)in the current string
not_number	next argument string can be any thing if it is True. Otherwise, next argument string has to be a numerical string.

Description:

Program Level:

kernel

Related Functions:

arget(), parse_argus()

Name:

bridge_header_handle — standard header handle interface

Synopsis:

```
bridge_header_handle(   int       job,
                        U_IMAGE   *img,
                        ...       )
```

Arguments:

job	header function name: HEADER_READ HEADER_FREAD HEADER_WRITE HEADER_FWRITE HEADER_FROM HEADER_TO HEADER_TRANSF ADD_DESC
img	image structure pointer
...	parameters. Details are in Description

Description:

All header routines return 0 on successful, or EOF (-1) on failure.

```
bool    (*img->header_handle)(HEADER_READ, img,
                        No, bool multi_frame, bool OsameI)

bool    (*img->header_handle)(HEADER_FREAD, img,
                        FILE* fp, bool multi_frame, bool OsameI)
```

Both HEADER_READ and HEADER_FREAD functions return 0 on successfully reading an image header and set up image structure for further processing according to the status set up by format_init(). Otherwise, return negative value.

multi_frame — for multi-frame non-HIPS image. If this True, look forward to reading information instead of seeking back.

OsameI — do not convert 24-bit color to 8-bit color image

```
bool    (*img->header_handle)(HEADER_WRITE, img,
                        int argc, char* argv, bool update_header);

bool    (*img->header_handle)(HEADER_FWRITE, img,
                        int argc, char* argv, FILE* fp);
```

It always update the header.

Both are or writing HIPS or FITS image header.

argc — number of arguments in argument list

argv — argument list

update_header — add all information in argv to header history field. Ignore argc and argv if this flag is False.

fp — file stream pointer for HEADER_FWRITE.

bool (*img->header_handle)(ADD_DESC, img, char* info)
Add info into header description field.

bool (*img->header_handle)(HEADER_TRANSF, img, No);
make output format and type same as input image's.

bool (*img->header_handle)(HEADER_FROM, img, struct header*);
bool (*img->header_handle)(HEADER_TO, img, bool initial_header);

Transfer other HIPS header to image structure, or image structure content to internal HIPS header. If struct header pointer is NULL in HEADER_FROM call, then internal header will be returned.

header	external HIPS header structure pointer
initial_header	clear internal HIPS structure content

Program Level:

interface

Related Functions:

(*img->header_handle)(), (*img->std_swif)()

Name:

buffer_close — destroy buffer structure created by buffer_create()

Synopsis:

```
void
buffer_close(BUFFER *bp)
```

Arguments:

bp	buffer pointer

Description:

buffer_close() will check buffer ID and release all buffer space

Return:

none

Program Level:

kernel

Related Functions:

buffer_create(), link_buffer()

Name:

buffer_create — create buffer structure for virtual file reading or writing

Synopsis:

```
buffer_create(int       buffer_size,
              int       header_size   )
```

Arguments:

buffer_size	buffer memory size. It can be zero and signed later.
header_size	buffer header size for holding any header structure

Description:

Return:

Program Level:

kernel

Related Functions:

Name:

buffer_read — read buffer

Synopsis:

```
buffer_read(char      *buf,
            int       block_size,
            int       num_blocks,
            BUFFER    *bp          )
```

Arguments:

buf	user buffer for reading data
block_size	data block size
num_blocks	number of blocks to be read
bp	buffer pointer

Description:

buffer_read() is similar fread. It reads data from a buffer pointer instead of a file pointer.

Return:

number blocks read or -1 on errors.

Program Level:

kernel

Related Functions:

buffer_create(), buffer_seek(), buffer_write()

Name:

buffer_seek — reposition buffer memory pointer

Synopsis:

```
buffer_seek(BUFFER    *bp,
            int       offset,
            int       from      )
```

Arguments:

bp	buffer pointer
offset	is an offset in bytes relative to the position specified by from
from	0 meaning 'beginning of the buffer memory' 1 meaning 'the current position' 2 meaning 'the end of the buffer memory'

Description:

Return:

adjusted position or -1 on errors.

Program Level:

kernel

Related Functions:

buffer_read(), buffer_write()

Name:

buffer_write — write data to a buffer

Synopsis:

```
buffer_write(char        *buf,
             int         block_size,
             int         num_blocks,
             BUFFER      *bp          )
```

Arguments:

buf	user buffer for reading data
block_size	data block size
num_blocks	number of blocks to be write
bp	buffer pointer

Description:

Return:

number blocks written or -1 on errors.

Program Level:

kernel

Related Functions:

buffer_create(), buffer_seek()

Name:

build_arg_fmt_list — * build system list from user arg_fmt_list_string

Synopsis:

```
arg_fmt_lists*
build_arg_fmt_list(     int                     *nitems,
                        arg_fmt_list_string*    arg_fmt,
                        va_list                 ap          )
```

Arguments:

Not suggest for user programming

Description:

interpret user arg_fmt_list_string to internal format for parse_argus() internal

Program Level:

kernel

Related Functions:

get_args()

Name:

build_arg_list — build an argv like argument list

Synopsis:

```
build_arg_list(  char*      list_string,
                 char**     arg_list_ptr[]    )
```

Arguments:

list_string a string contains many options, such as a command line string

Description:

build_arg_list() splits a string, including white space, to many single item, no including any white space.

Program Level:

kernel

Related Functions:

get_arg_list()

Name:

build_ccs_table — allocate CCS table content space

Synopsis:

Arguments:

Description:

Return:

Program Level:

kernel

Related Functions:

ccs_table_if()

Name:

build_socket — create socket I/O stream for different network application

Synopsis:

```
build_socket( char*            host,
              char*            serv_o_port,
              int              sock_type,
              bool             dir,
              int              *buf_size,
       struct sockaddr_in      *bsin,
              magic_server     *ms          )
```

Arguments:

host	host name or IP address for transmission
serv_o_port	server name or port number string
sock_type	SOCK_type. See sys/socket.h
dir	True or ~True for transmitter, False and ~False for receiver. True and False are simplex mode. The negative values are duplex mode (~False = -1, ~True = -2)
buf_size	window size for TCP or buffer size for UDP. If given size is too big, the kernel will automatically find the suitable one and return it by this pointer, so be sure to pass it by pointer rather than value.
bsin	socket address information structure pointer. NULL is ok.
ms	socket information return pointer. NULL is ok for no request.

```
typedef struct {
        int              as, so, /* accepted socket & original socket fd */
                         connected, m_port,
                         dp, stat,
                         total, packet_size;
        char             *bp;
        struct timeval   ts, te;
        } magic_server;
```

Description:

build_socket() is designed to fit different network socket requirement. Once a socket is created, *build_socket()* will check window (buffer) size and bind the socket in a proper way. After calling *build_socket()*, you can set socket options when you need. For TCP applications, the accept or connect is the only job before read or write, respectively. No extra job for UDP applications.

Program Level:

kernel

Related Functions:

socket_connect()

Name:

bytes_in_colortype — get number of bytes for given color type.

Synopsis:

bytes_in_colortype(int color_type)

Arguments:

color_type image color type. It is defined in imagedef.h

Description:

bytes_in_colortype() returns number of bytes used for each color cell.

Program Level:

kernel

Related Functions:

Name:

calibrate_colors — rearrange the color map according to its frequency

Synopsis:

alibrate_colors(U_IMAGE *img)

Arguments:

img	image structure pointer

Description:

alibrate_colors() computes color cell frequencies in color map and rebuilds color map according to frequencies. In other words, it puts the highest frequent color cell in the first entry in the color map, and the least frequent color cell in the last entry. Therefore, the color allocating routine can easily allocate available color cell for the most frequently used color and left the least frequently used color to use the closest color in existing color map if there is no enough color cell for every color entry.

Program Level:

kernel

Related Functions:

To_8(), quant_to_8(), q_sort(), cmap_t* reg_cmap[3]

Name:

ccs_get_row —input image line by line

Synopsis:

ccs_get_row(U_IMAGE *img)

Arguments:

img image structure pointer

Description:

Usually, kernel will input entire frame and send to applications. If applications want to input an image row by row (line by line), it can use this routine to do so. Note: use *ccs_get_row_ok()* function call to determine if it is feasible since some image format can not be input line by line, such as separated plane format. Some of these format, however, may be handled virtually by kernel, so see if *ccs_get_row_ok()* or not before use *ccs_get_row()*.

Program Level:

kernel

Related Functions:

ccs_get_row_ok()

Name:

ccs_table_if — CCS internal table interface

Synopsis:

```
ccs_table_if(int          job,
             U_IMAGE      *img,
             va_list      ...          )
```

Arguments:

Description:

see (*img->table_if)()

Return:

Program Level:

kernel

Related Functions:

Name:

check_host — function returns host type ID

Synopsis:

check_host()

Arguments:

N/A

Description:

check_host() returns a host type ID:

0	word = 8-bit machine
1	word = 16 bits, little endan
2	word = 16 bits, big endan
3	word = 32 bits, little endan
4	word = 32 bits, little endan
5	word = 32 bits, big endan
6	word = 64-bit, little endan
7	word = 64-bit, big endan machine

Program Level:

user

Related Functions:

Name:

color_rotate_90 — rotate a color image in 90 degree

Synopsis:

```
color_rotate_90(VType    *in_buf, *o_buf,
                int      new_height, new_width,
                int      color_form, rotate_left     )
```

Arguments:

in_buf	original image buffer
o_buf	rotated image buffer
color_form	image color format
rotate_left	rotate counter-clockwise if true; otherwise, rotate clockwise

Description:

color_rotate_90() rotates color image data in in_buf and output to o_buf. The color_form is defined in imagedef.h (CFM_????).

Program Level:

kernel

Related Functions:

ilc_transfer(), rotate90()

Name:

comfirm_host — **

Synopsis:

```
comfirm_host(U_IMAGE    *img,
             int        host_type,
             bool       isFORTRAN_FILE)
```

Arguments:

Description:

Program Level:

kernel

Related Functions:

Name:

core_trace — **

Synopsis:

core_trace(...)

Arguments:

Description:

Program Level:

kernel

Related Functions:

Name:

create_pr_colormap — * allocate Raster color map space

Synopsis:

```
colormap_t*
create_pr_colormap(void     )
```

Arguments:

None

Description:

create a Raster color map space with RMT_EQUAL_RGB type. The color map content is empty (zeros).

Program Level:

kernel

Related Functions:

free_pr_colormap(), set_pr_colormap()

Name:

dump_tbl — dump any table with LKT type

Synopsis:

```
dump_tbl(    LKT     *table,
             int     table_size,
             int     display_cols,
             char    *table_name    )
```

Arguments:

table	LKT type table pointer
table_size	how big is the table
display_cols	how many columns on each page. e.g. 2 columns = 123.00 345.67 3 columns = 123.000 345.67 678.89
table_name	the name or title appears on the printing page

Description:

Program Level:

kernel

Related Functions:

Name:

dvfft
dvfft_2d
dvfftn
dvrft_2d — Virtual Fast Fourier Transform

Synopsis:

```
dvfft(      double   *re_plane,
            double   *im_plane,
            int      len_in_log2     )

dvfft_2d(   double   *re_plane,
            double   *im_plane,
            int      rows_ing_log2,
            int      cols_in_log2    )

dvfftn(     double   *re_plane,
            double   *im_plane,
            int      len_in_log2,
            int      span_len        )

dvrft_2d(   double   *re_plane,
            double   *im_plane,
            int      rows_in_log2,
            int      cols_in_log2    )
```

Arguments:

Description:

Program Level:

kernel

Related Functions:

Name:

dw_init
dw_load — double VFFT initialization routines

Synopsis:

dw_init(	MType	half_len	)
dw_load(	int	points	)

Arguments:

Description:

not suggested for user program

Program Level:

fuzzy (soft-tissue)

Related Functions:

vfft

Name:

error_mesg — error message reporter

Synopsis:

error_mesg(void)

Arguments:

N/A

Description:

error_mesg() reports an error and its level if error happens

Program Level:

kernel

Related Functions:

message(), prgmerr(), syserr()

Name:

eta_curve — Elastic Tuning curve generator

Synopsis:

eta_curve(	LKT	*lkt,	
	float	scale,	
	int	max_diff,	
	int	type	)

Arguments:

lkt	histogram lookup table
scale	ETA scale factor
max_diff	top and bottom value difference in histogram
type	foreground <1> or background

Description:

Program Level:

kernel

Related Functions:

histogram_calc()

Name:

exit_timeout — * exit when license time out

Synopsis:

exit_timeout(	char	*message,	
	int	year, month, day, hour, min	)

Arguments:

message	exit message
...	date

Description:

Program Level:

assemble

Related Functions:

Name:

find_input_type — * find input type for given argument format

Synopsis:

```
find_input_type(char**    input_format_ptr,
                def_val   def_val,
                bool      set_val,
                convs     *conversion_cell,
                va_list   *ap               )
```

Arguments:

input_format_ptr	pointer to a position in an argument input format string
def_val	default value (or pointer if USE_STRING_ARGU_DEF compiling directive is defined for compiling CCS library)
set_val	set default value to corresponding variable
conversion_cell	conversion structure pointer in argument format structure.
ap	variable list pointer

Description:

Program Level:

kernel

Related Functions:

build_arg_fmt_list(), parse_argus()

Name:

fits_convertdata — *

Synopsis:

```
float
fits_convertdata(U_IMAGE *img,
                 long     *in_buf,
                 long     length,
                 VType    *o_buf         )
```

Arguments:

img	image structure pointer
in_buf	original data buffer
length	data size in bytes
o_buf	converted data buffer

Description:

This procedure swaps and converts data only from fits format to regular data format between big-endan and little-endan machine. It uses two global variables: hostype and FTy. The hostype gives current machine type, and FTy gives type of machine which generated and FITS data. The FTy is input from command line or given by hand. The hostype is set by check_host() function.

Program Level:

kernel

Related Functions:

check_host(), int FTy;

Name:

fits_header_handle — * FITS header handle interface

Synopsis:

```
fits_header_handle(    int        job,
                       U_IMAGE    *img,
                       ...        )
```

Arguments:

Description:

Program Level:

interface

Related Functions:

bridge_header_handle(), (*img->header_handle)()

Name:

fits_transf_data — * FIST data transfer kernel

Synopsis:

```
fits_transf_data(U_IMAGE  *img,
                 bool      file_type,
                 bool      write_flag    )
```

Arguments:

img	image structure pointer
file_type	is a FORTRAN file
write_flag	write data to img->OUT_FP; otherwise, convert data to img->src buffer

Description:

Program Level:

kernel

Related Functions:

fits_convertdata(), read_var(), read_fits_image()

Name:

fits_uncompress — * FITS image compression kernel

Synopsis:

```
fits_uncompress(U_IMAGE*img,
                bool        file_type        )
```

Arguments:

Description:

Program Level:

kernel

Related Functions:

Name:

format_init — image structure initiate

Synopsis:

```
format_init(    U_IMAGE  *img,
                int      in_type, mid_type, out_type,
                char     *caller_name, *version          )
```

Arguments:

img	image structure pointer
in_type	input type. See description for detail
mid_type	middle type (converting result)
out_type	output type for writing image
caller_name	program name for error indicating
version	program version

Description:

format_init() initializes image structure (U_IMAGE) with proper input, conversion and output image types, color format, and I/O handlers. The *in_type* is suggested to use IMAGE_INIT_TYPE macro unless you want to input a special image type and ignore others. It is a hierarchy for the CCS internal library. That is, if IMAGE_INIT_TYPE is defined as RAS, then the CCS internal library will not look at GIF, RLE, FITS and HIPS type for input (see imagedef.h). The *mid_type* is an image type to which the input will be converted. The HIPS and RLE will cove almost all image types. So, use RLE if RLE is a desired converting format; otherwise, use HIPS for all other converting formats. The *out_type* is used for writing image out, the available types are HIPS, RLE, and RAS. The caller_name and version are used for error handling.

The conversion type, color format and image display channels can be changed after reading the headers. These fields are:

img->mid_type, img->color_form, and img->dpy_channels

These fields should be set up before calling the data read routine: (*img->std_swif)(FILE_LOAD, img, ...). Be sure that when these parameters are changed, the color format 'img->color_form' must agree with img->dpy_channels; otherwise, the conversion may not be properly done. Once, these variables are set up, the image read handler will automatically convert input image to that image type and color format. The available color conversion formats are:

CFM_SGF	gray scale
CFM_SCF	8-bit color (pseudo color)
CFM_ILC	interleave cell (RAS)
CFM_ILL	interleave line (RLE)
CFM_ALPHA	RAS with an alpha channel
CFM_SEPLANE	separate plans

format_init() initializes gray scale image format by default. Color image format can be initialized if img->color_dpy is set to True (non-zero) before *format_init*() is called.

Program Level:

interface

Related Functions:

bridge_header_handle(), (*img->header_handle)(), (*img->std_swif)()

Name:

free_2d_discrete — free a 2-dimensional memory space

Synopsis:

```
ree_2d_discrete(VType*    p,
                int       rows        )
```

Arguments:

p	pointer allocated by *alloc_2d_discrete()* subroutine
rows	number of rows

Description:

free_2d_discrete() frees memory allocated by *alloc_2d_discrete()*

Program Level:

kernel

Related Functions:

alloc_2d_discrete()

Name:

free_arg_fmt_list — * free an argument format list

Synopsis:

```
free_arg_fmt_list(          int               n_items,
                            arg_fmt_lists*    fmt_list          )
```

Arguments:

n_items	number of flags in the argument format list to be freed
fmt_list	argument format list

Description:

free_arg_fmt_list() free all memory space allocated by *build_arg_fmt_list*().

Program Level:

kernel

Related Functions:

build_arg_fmt_list()

Name:

free_ccs_table — release CCS table content space

Synopsis:

```
void
free_ccs_table(ccstable *ct )
```

Arguments:

ct	CCS table header pointer

Description:

Return:

Program Level:

kernel

Related Functions:

build_ccs_table(), ccs_table_if()

Name:

free_pr_colormap — free color map created by create_pr_colormap

Synopsis:

```
void
free_pr_colormap(colormap_t* pr_cmap)
```

Arguments:

pr_cmap	Sun color map pointer

Description:

free_pr_colormap() frees a color map structure created by create_pr_colormap()

Return:

Program Level:

kernel

Related Functions:

create_pr_colormap()

Name:

gauss_mask — generate Gauss masks

Synopsis:

```
double
gauss_mask(   double     sigma,
              int        num_mask,
              Float      *mask_array,
              int        precision,
              bool       gen_gaussonly  )
```

Arguments:

sigma	sigma. Must be non-zero
num_mask	mask array size (window size)
mask_array	linear Float array (results)
precision	deviation differential ratio. Must be a positive number.
gen_gaussonly	print Gauss masks to standard output

Description:

gauss_mask() generates a set of Gauss masks, prints them to standard output if gen_gaussonly flag is given, and returns 1/SUM(masks).

Program Level:

assemble

Related Functions:

Name:

get_addr — get network IP address in long integer

Synopsis:

```
get_addr(    char     *hostname,
             long     *addr_op     )
```

Arguments:

hostname	either name or IP string
addr_op	return address

Description:

get_addr() translates hostname string to a long integer

Program Level:

kernel

Related Functions:

build_socket(), get_port()

Name:

get_arg_list — get argument format list

Synopsis:

get_arg_list(VType *ret_ptr)

Arguments:

ret_ptr return pointer for argument list generated by *build_arg_list()*; it should be the address of a char pointer array, such as, char **argp -> &argp.

it also returns number of arguments in the list.

Description:

Program Level:

assemble

Related Functions:

build_arg_list()

Name:

get_args — * get argument lists

Synopsis:

```
get_args(      int              n_arg_in_list,
          arg_fmt_lists          *fmt_list,
               int               argc,
               char**            argv,
               int               *cur_n,
               int               field              )
```

Arguments:

n_arg_in_list	number flags in argument format list
fmt_list	argument format list
argc	number argument in command line argument list
argv	command line argument list
cur_n	current position pointer to argv list
field	current character in current argv list

Description:

Program Level:

kernel

Related Functions:

Name:

get_fits_head — * FITS header handle kernel

Synopsis:

```
get_fits_head(  FITS_BASE*fits_hd,
                U_IMAGE*img,
                int         host_type,
                bool        isFORTRAN_FILE          )
```

Arguments:

Description:

Program Level:

kernel

Related Functions:

bridge_header_handle(), (*img->header_handle)(), check_host()

Name:

get_infile — create a readable file by name

Synopsis:

```
FILE *
get_infile(     int      host_type,
                char     *file_name,
                char     *mesg        )
```

Arguments:

Description:

get_infile() open a file as read only by given *file_name*.

Program Level:

assemble - input/output

Related Functions:

get_outfile()

Name:

get_outfile — create a writable file by name

Synopsis:

```
FILE *
get_outfile(    int     host_type,
                char    *file_name,
                char    *mesg       )
```

Arguments:

Description:

Program Level:

assemble

Related Functions:

get_infile()

Name:

get_port — get network port for transmission

Synopsis:

```
get_port(    char    *server_name_or_port,
             bool    udp                  )
```

Arguments:

server_name_or_port	either server name or port number
udp	False for TCP

Description:

Program Level:

kernel

Related Functions:

build_socket(), get_addr()

Name:

get_soaddr — get socket information of connection.

Synopsis:

```
struct sockaddr_in *
get_soaddr( int                 which,
        struct  sockaddr_in*  asin          )
```

Arguments:

which	1 for transmitter socket info, and 0 for receiver's or transmitter's.
asin	socket connection information

Description:

get_soaddr() returns the internal sockaddr_in structure filled by build_socket(). If asin is a non-NULL pointer, get_soaddr() will copy the internal sockaddr_in structure into asin space and returns the address; otherwise, only the address is returned.

Program Level:

kernel

Related Functions:

build_socket(), get_addr(), rtp_open()

Name:

get_superimpose_param — *

Synopsis:

```
get_superimpose_param(U_IMAGE*        img,
                      struct header*  hips_hd      )
```

Arguments:

Description:

Program Level:

kernel

Related Functions:

Name:

getbyte — **

Synopsis:

getbyte(FILE *fp)

Arguments:

fp file stream pointer

Description:

Program Level:

fuzzy

Related Functions:

Name:

gif_header_handle — * GIF header handle interface

Synopsis:

```
gif_header_handle(int       job,
                  U_IMAGE   *img,
                  ...       )
```

Arguments:

Description:

Program Level:

interface

Related Functions:

bridge_header_handle(), (*img->header_handle)()

Name:

gray_to_rle — * RLE gray scale image to ILL (RLE) format convertor

Synopsis:

```
gray_to_rle(     byte           *o_buf, *in_buf,
                 U_IMAGE        *img,
                 int            colormap_len,
           unsigned short       colormap         )
```

Arguments:

o_buf	output buffer pointer
in_buf	input buffer contains data
img	image structure pointer
colormap_len	RLE color map length
colormap	RLE color map

Description:

gray_to_rle() only for converting RLE gray scale image to ILL format.

Program Level:

fuzzy

Related Functions:

Name:

hips_header_handle — * HIPS header handle interface

Synopsis:

```
hips_header_handle(int      job,
                   U_IMAGE  *img,
                   ...      )
```

Arguments:

Description:

Program Level:

interface

Related Functions:

bridge_header_handle(), (*img->header_handle)()

Name:

histogram — computer histogram

Synopsis:

```
histogram(   byte      *buf,
             int       size,
             int       *hist_buf,
             U_IMAGE   *img         )
```

Arguments:

buf	data buffer
size	buffer size in bytes
hist_buf	histogram buffer pointer
img	image structure pointer

Description:

Program Level:

kernel

Related Functions:

Name:

histogram_calc — histogram calculation

Synopsis:

```
histogram_calc( byte    *buf,
                int     size,
                int     *hist_buf    )
```

Arguments:

buf	buffer containing data
size	buffer size in bytes
hist_buf	histogram buffer pointer

Description:

Program Level:

kernel

Related Functions:

Name:

icc_header_handle — * ICC header handle interface

Synopsis:

```
icc_header_handle(int      job,
                  U_IMAGE  *img,
                  ...      )
```

Arguments:

Description:

Program Level:

interface

Related Functions:

bridge_header_handle(), (*img->header_handle)()

Name:

if_time_exp — * check if license time expired

Synopsis:

if_time_exp(time_t tm_in)

Arguments:

tm_in	current time

Description:

Program Level:

assemble

Related Functions:

Name:

ilc_to_gray — ILC (Raster) color image to gray scale image convertor

Synopsis:

```
ilc_to_gray(    byte*      out,
                byte*      rgb,
                int        n_elems,
                bool       alpha,
                cmap_t*    colormap[],
                bool       Regular_T      )
```

Arguments:

out	output buffer pointer
rgb	input ILC buffer pointer
n_elems	number pixel need to be converted
alpha	if input has alpha channel (ARGB or RGBA)
colormap	if input is 8-bit color image
Regular_T	RGB or BGR

Description:

Program Level:

kernel

Related Functions:

map8_gray()

Name:

ill_to_gray — ILL (RLE) or SEPLANE image to gray scale image format convertor

Synopsis:

```
ill_to_gray(    byte*    out,
                byte     *r_p, *g_p, *b_p,
                int      line_width               )
```

Arguments:

out	output buffer pointer
r_p, g_p, b_p	scanline or frame pointers for input
line_width	scanline width

Description:

ill_to_gray() can convert ILL and SEPLANE images to gray scale image. For ILL image conversion, it does line by line. For SEPLANE image, it can do region by region or entire frame.

Program Level:

kernel

Related Functions:

ilc_to_gray(), map8_gray()

Name:

ilc_to_sep— ILC (RAS) to SEPLANE image convertor

Synopsis:

ilc_to_sep(	byte	*out, *rgb_in,	
	int	size, channels,	
	bool	reverse	)

Arguments:

out	output buffer pointer
rgb_in	input buffer contain ILC image data
size	pixels will be converted
channels	input image channels
reverse	reverse RGB channels to BGR channels

Description:

Program Level:

kernel

Related Functions:

Name:

ilc_transfer — ILC format converter

Synopsis:

```
ilc_transfer(   char    *obp, *ibp,
                int     width,
                int     in_channels,
                bool    reverse,
                int     out_channels    )
```

Arguments:

obp	output buffer pointer
ibp	input buffer pointer
in_channels	channels in an input image
reverse	reverse RGB to BGR or another way around
out_channels	channels in an output image

Description:

ilc_transfer() converts RGB, ARGB, BGR or BGRA ILC image to one of other format.

Program Level:

kernel

Related Functions:

Name:

ill_to_sep — ILL (RLE) to SEPLANE image converter

Synopsis:

```
ill_to_sep(   byte    *out, *ill_in,
              int     width, height,
              int     channels      )
```

Arguments:

out	out buffer to store SEPLANE data
ill_in	buffer contains ILL input data
width, height	scanline width and number of lines
channels	color channels

Description:

ill_to_sep() can convert entire ILL image to a SEPLANE image, or part of ILL image to a new smaller SEPLANE image.

Program Level:

kernel

Related Functions:

Name:

(*img->errors)()
(*img->header_handle)()
(*img->read)()
(*img->seek)()
(*img->std_swif)()
(*img->write)()
(*img->table_if)()
(*img->map_scanline)()
(*img->MAG_scanline)()
U_IMAGE internal function pointers — interface handler

Synopsis:

```
(*img->errors)( int          error_level,
                char         *fmt_msg,
                ...                         )

(*img->header_handle)(       int           header_job_id,
                             U_IMAGE       *img,
                             ...                          )

(*img->read)(   VType        *buf,
                int          nblock,
                int          block_size,
                FILE         *fp            )

(*img->seek)(   FILE         *fp,
                int          offset,
                int          where          )

(*img->std_swif)(            int           job_id,
                             U_IMAGE       *img,
                             ...                          )

(*img->write)(  VType        *buf,
                int          nblock,
                int          block_size,
                FILE         *fp            )

(*img->table_if)(int         table_job_id,
                U_IMAGE      *img,
                ...                         )

(*img->map_scanline)(        Image         *img,
                             byte          *rgb[3],
                             int           ncolors,
                             int           map_windth, stride,
                             int           y, x,
                             XImage        *image         )
```

```
(*img->MAG_scanline)(   Image       *img,
                        int         x0, y0,
                        int         mag_fact,
                        int         x, y,
                        int         width, height,
                        Ximage      *image      )
```

Arguments:

(*img->errors)():

error_level — error level. A positive number is fatal and causes program to terminate. Level 0 reports errors and return EOF. Negative level will return ~level. See prgmerr()

(*img->header_handle)(): see bridge_header_handle()

(*img->read)(): same as fread in C by default. It is used for handling the pipe seeking when system determines the OS can not handle pipe seek, or set by macro init_readpipe(U_IMAGE *img).

(*img->seek)(): same as fseek() in C by default. Refer to (*img->read)().

(*img->std_swif)(): see std_interface()

(*img->write)(): same as fwrite() in C by default. Refer to (*img->read)().

(*img->table_if)(): standard table interface (not available yet)

(*img->map_scanline):

img	image structure pointer
rgb	1 - 3 buffer pointer
ncolors	number of image channel
map_width	scanline width to be mapped. It can be part or entire line.
stride	1 for continuous map, 2 or more for spanning map.
y	current line position (yth row)
x	map starting column in current row. x + map_width must less than the image width (scanline width)
image	Ximage structure pointer to display the mapped image

(*img->MAG_scanline):

img	image structure pointer
x0, y0	magnify origin
mag_fact	magnify factor (negative number is shrinking)
x, y	destination origin to X image data buffer (window)
width, height	window size to be changed
image	Ximage structure pointer to display the magnified image

Description:

The table interface is designed to use known decode routines in CCS kernel for new image types. The image header can be either read or skipped according to a description in table files. This gives users a way to quickly add a new conversion function without increasing program size and invoking programming time. To active the internal table interface, set up the env variable CCSTABLE_LIST to the file contains a list of table files. e.g.

```
setenv CCSTABLE_LIST ~/table.dir/ccstable.lst
```

The ccstable.lst should contains only names of table files.

```
demo% more ~/table.dir/ccstable.lst
# this file contains a list of conversion table file names
~/table.dir/table.vff
/tmp/table.demo

demo% more ~/table.dir/table.vff
header  4  161  anim    skip
width    0    #    128
height    0    #    128
frames  0    #    32
depth    0    #    8
colormap  256    DEF
format  0    #    SCF
encode  0    #    RAS
```

The detailed table format is in Chapter 3.

Program Level:

interface

Related Functions:

format_init()

Name:

init_FS_tables — * initialize Floyd-Steinberg tables for quantization

Synopsis:

void
init_FS_tables(cmap_t *colormap[])

Arguments:

colormap color map pointer to hold color map generated for dithering 24-bit color image to 8-bit color image

Description:

init_FS_tables() initializes a Floyd-Steinberg table for color dithering

Program Level:

kernel

Related Functions:

To_8()

Name:

isColorImage — check if the format represent a color image

Synopsis:

isColorImage(int color_form)

Arguments:

color_form color image format CFM_????

Description:

isColorImage() returns color image channels or 0 for gray scale images.

Program Level:

kernel

Related Functions:

Name:

jpeg_header_handle — * JPEG image header handle interface

Synopsis:

```
jpeg_header_handle(int       job,
                   U_IMAGE   *img,
                   ...       )
```

Arguments:

Description:

Program Level:

kernel

Related Functions:

Name:

last_pointer_pos — get last memory pointer position

Synopsis:

last_pointer_pos(VType *p)

Arguments:

p allocated pointer

Description:

currently, this function only implemented on PC by DOS kernel.

Program Level:

kernel

Related Functions:

Name:

line_to_cell_color — ILL (RLE) to ILC (RAS) image converter

Synopsis:

```
line_to_cell_color(char     *obp, *r_rle,
                   int      width, height   )
```

Arguments:

Description:

Program Level:

kernel

Related Functions:

Name:

line_to_sep_color — ILL (RLE) to SEPLANE image converter

Synopsis:

```
line_to_sep_color(char    *obp, *r_rle,
                  int     width,
                  int     height,
                  int     channels     )
```

Arguments:

Description:

Program Level:

kernel

Related Functions:

Name:

link_buffer — set buffer handle routines to U_IMAGE structure

Synopsis:

```
link_buffer(U_IMAGE      *img,
            int          job          )
```

Arguments:

img	image structure pointer
job	not used

Description:

link_buffer() assigns buffer handle routine to img structure to substitute the file handle routine, such as fread(), fseek(), ... etc.

Return:

Program Level:

kernel

Related Functions:

buffer_create()

Name:

load_DBw, lload_w — * Double FFT initial routines

Synopsis:

load_DBw(	unsigned	elems	)
load_w(	unsigned	elems	)

Arguments:

Description:

Program Level:

kernel

Related Functions:

Name:

load_rastfile — * load Sun raster files

Synopsis:

```
load_rastfile(     byte              *buf,
              struct rasterfile  *rast_hd,
                   int               size,
                   FILE              *fp          )
```

Arguments:

Description:

Program Level:

assemble

Related Functions:

Name:

map8_gray — map 8-bit color image to gray scale image

Synopsis:

```
map8_gray(   byte      *out,
             byte      *in_8,
             int       len,
             cmap_t    *colormap[3]   )
```

Arguments:

Description:

Program Level:

kernel

Related Functions:

Name:

message — print information to stderr channel

Synopsis:

```
message(    char*      mesg_format,
            va_list    mesg_list       )
```

Arguments:

mesg_format	format string
mesg_list	messages

Description:

Program Level:

kernel

Related Functions:

error_mesg(), prgmerr(), syserr()

Name:

min_bits — minimum bits for represent a number

Synopsis:

min_bits(int val)

Arguments:

val	a number

Description:

min_bits() returns the minimum bits for storing a number which is given by *val*.

Program Level:

kernel

Related Functions:

Name:

nzalloc — allocate non-clean memory

Synopsis:

VType*			
nzalloc(	int	blocks,	
	int	size,	
	char*	caller_name	)

Arguments:

blocks	number of blocks
size	block size
caller_name	the calling routine name or any name for reference

Description:

nzalloc() allocates memory space with size = *blocks* * *size*. All memory handle functions and subroutines are tied to kernel debug, so once the global variable *"debug"* is turned on (any level), then debug will print all memory information whenever memory allocating status are changed.

In case of memory allocation failed, *nzalloc()* prints error message, and exits from system if the caller name is given, or return NULL if the caller name is NULL or No.

The macro for *nzalloc()* is NZALLOC() which is less overhead.

Program Level:

kernel

Related Functions:

pointer_buffer_size(), verify_buffer_size(), last_pointer_pos(), zalloc()

Name:

parse_argus — parse argument list for command handling

Synopsis:

```
parse_argus(char*              *file_list[],
            int                argc,
            char               **argv,
     arg_fmt_list_string*      arg_fmt_list,
            va_list            va_alist          )
```

Arguments:

file_list	a pointer points to a double file list array (pointer)
argc	numbers of input arguments in argv
argv	input argument lists
arg_fmt_list	argument format and information lists
va_alist	a list of argument variables

Description:

parse_argus() resolves elements in argv into a list of argument values according to arg_fmt_list, and returns number of files in argv. If something in argv list does not match anything in arg_fmt_list, it print usage information and return EOF.

Argument Format List:

flag-string	input-format	default value	number of variables	mini-mum inputs	infor-mation string	scan flag

example:

```
{"-f",     "%f",   3.14159,     1,          1    "scale factor"     }

{"This is information only",
           NULL,     0,          0,          0,  "End of usage"}
```

The last format list must be a NULL list. See examples below.

Flag format and its extension:

It can be any string which you want to use in your program. Regularly, it looks like:

"-d", "-scale", or "-s[cale], even "-f[x][y], and "+bar"

The flag in a pair of square brackets is called ***flag extension***. In extension examples above, it means you can type -s or -scale, and you can type -f -fx -fy for different or same variable.

The special flag "-#" means -number, such as -1.2 --3.4 -0x7F. The important thing is that this flag must be at the first entry in the argument format list. The # symbol can also be an extended flag, such -F[#}, this extension will accept two variables in is list. The first variable is associated with flag -F, and the second variable is associated with flag -F# , -F[0-9], or -F [0-9]. So, in this case, the first variable is better a non-digital variable.

See program examples below.

Input Format:

Similar to C language format, so user can easily use it.

%c character.
%d %i integer.
%h %S short integer.
%x %X hex integer.
%f floating point.
%b boolean format.
%g %D double floating point format.
%s string.

Additional boolean formats:

%0 - %31 set variable to value 0 - 31 when flag is given.
%! reverse variable content when flag is given.
%~ set variable to 0 if next input is numerical; otherwise set it to 1.
%+ %- increase or decrease variable value by one if flag is given.
%# set variable to True if next input is a number. Opposite %~ format operation.
%& %| %^ %* logic operation with default value if flag is given.
%B return extended argument type to variable when flag is given.
%E return extension value to variable when flag is given.
%N reset variable value to default value when flag is given.
%?h ? is any operation above. Any operation is in short format. Default is in long operation format.

These boolean formats can be used with other formats by putting it at the beginning of the format list to determine if the input flag is appeared and set a special boolean value, or used alone. Every *extension flag* will effect these boolean values, if a *flag* has extensions.

If format is NULL string or an empty string, then, this format list will be treated as information only, it is like second example in **Flag Format** description.

Default Value:

The default value is used for two ways: initialization and set value.

<1> Initialization only happens when input format is non-boolean format, such as, character, integers, floating point (double), and the input value is enforced (Minimum Number of Inputs).

<2> The cases other than case <1> and flag is given.

e.g.:

```
int         rows, cols=64, frames=8;
arg_fmt_list_string        arg_fmt[] =        {
            {"-window", "%d %d %d", 128, 3, 1, "[y x z] output window size"},
            NULL               };

parse_argus(NULL, argc, argv, arg_fmt, &rows, &cols, &frames);
```

When we give -window 256, the rows = 256, and cols = frames = 128.
If we run it without option, then rows = 128, cols = 64, and frames = 8.

See program examples for more information.
See Information String for display this value.

Number of Variables:

The number of variables should match the number of input formats. e.g., if input format is "%s %d %f", then the number of variables is three. The flag extension, however, can have only value one for this field, even though there are still three variables in argument variable list. e.g., "-f[x][y] . It is because the main variable is one. The others are extensions.

For none flag entry (comments or information), this field must be zero.

This field can be dynamically generated from input flag and format. Therefore, this field can be taken out of arg_fmt_list_string structure to save space and programmer's work. However, two reason makes this field to live:

<1> reduce kernel program complexity.

<2> help programmer to remember how many variables need to be put in argument list.

Minimum Number of Inputs

This is the option to force input a value. It means that there must be at least number of inputs required for non-boolean variables.

If input format is a string, then, this field means to do initializing on first n string pointers (positive number) or not (0). Otherwise, If this field is zero, then the first variable for this format must be a boolean variable.

Information String:

This field can be a NULL string if you have no information to tell. It is suggested to have this field to format a nice manual page when errors happen. The new line '\n' and tab '\t' are used at beginning of strings to adjust its position.

The "%f" format can be embeded in an information string to display the deafule value. e.g.:

{"-size", DefaultSize, 1, 1, "buffer size for input (default size = %.f)"}

Scan Flag:

This field tells kernel to use scanf() function instead of atoi(). This is nice way to prevent that the program changes you variable's default value. In example in the ***Default Value*** section, when input flag is "-window 256", the values of variables "cols" and "frames" were changed to the given default value 128. If you set ***Scan Flag*** to True, then this would not happen. Remember that you need to quota the multiple arguments on some machine. Ignore this field if you do not care it, unless you put an arg_fmt_list_string in a subroutine, then this field need to be set to zero.

Program Examples:

Design an argument format list for ***scale_geom*** filter:

Functionality and their	*input flags* and	*formats*:
scale factor	"-#"	floating point
output color image	"-c"	boolean
dimensions (single)	"-dc" "-dr" "-df"	integers
dimensions (group)	"-d"	integers
filters	"-filt"	strings
headers for pure data input	"-H"	boolean & integers
source box	"-s"	boolean & integers
supported radius in filters	"-supp"	double floats
x -> y or y -> x filtering	"-xy" "-yx"	boolean
input format for pure data	"-B" "-F" "-S"	value booleans
input and output information	None	none
end		

Entry:

```
[1]   {"-#", "%f", 1.0, 1, 1,
            "image will be shrinked or enlarged # times (-5.0 ~ +5.0)\n\
            # means '-real_number', such as: -1.1, --4.75, ..."}
[2]   {"-c", "%b", True, 1, 0, "output color (RLE) image"}
[3]   {"-d[c][r][f]", "%d %d %d %d", 0, 1, 1,
               "number of cols, [rows], [frames] in destination file\n\
               (default = source size)"}
[4]   {"-D", "%d %d %d", 0, 3, 2,
               "number of cols, rows, [frames] in file dest file"}
[5]   {"-filt", "%s %s %s", No, 3, 0, "\n\
               filter name in x, y, and z (default = triangle)\n\
               \"-filt '?'\" prints a filter catalog"}
[6]   {"-H", "%~ %d %d %d", 0, 4, 0,
         "\n\t[\"]row [column [frame]\"] specify pure image input size",
               True}
[7]   {"-s", "%# %d %d %d %d", 1, 5, 4,
               "source box (row col #rows #cols)"}
[8]   {"-supp", "%g %g %g", -1, 3, 1, "filter support radius"}
[9]   {"-xy", "%b", True, 1, 0, "filter x before y"}
[10]  {"-yx", "%b", False,1, 0, "filter y before x"}
[11]  {"-B", "%N", IFMT_BYTE, 1, 0,
               "input is BYTE formatted image"}
[12]  {"-F", "%N", IFMT_FLOAT, 1, 0,
               "input is FLOAT formatted image"}
[13]  {"-S", "%N", IFMT_SHORT, 1, 0,
               "input is SHORT formatted image"}
[Inf]   {" [< ] in_file [> | ] out_file", NULL, 0, 0, "end of information"}
```

Explanation:

entry[1] has to be in the first entry.

entry[3] contains 1 major flag and 3 extensions: -d with -dc -dr and -df

entry[4] since flag “-d” is used for entry[3], so “-D” is the choice for this entry.

in entry[6], the format “%~” means that once“-H” option is given on command line, the boolean value will be True if there is no any number following, or False if there is a set of number(s) following. The last field in entry[6] is True. It tells kernel do not change the variables’ values if no value follows the “-H” option flag; and on most machine, all the values must be quoted in a pair of double quotation makes.

in entry[7], the format “%#” is opposite the format “%~” in entry[6]. It means that if there is number following the option flag “-s”, set boolean variable to True, otherwise, set boolean variable to False. Also to remember that entry[8] must put after entry[7] because parsae_argus() requires alphabet order for similar flags, such as “-a”, “-ab”, and “-abc”, etc.

entry[9] and [10] set same variable “dir_xy” to opposite value.

entry[11 - 13] set same variable to different default value.

entry[Inf] is an optional entry.

Here is the ***real program*** looks like:

```
/* scale_geom . c
*   copyright(c)    LBL
*/
#include “header.def”

arg_fmt_list_string   arg_fmts[] =   {
    {“-#”, “%f”, 1.0, 1, 1,
                “image will be shrinked or enlarged # times (-5.0 ~ +5.0)\n\
                # means ’-real_number’, such as: -1.1, --4.75, ...”},
    {“-c”, “%b”, True, 1, 0, “output color (RLE) image”},
    {“-d[c][r][f]”, “%d %d %d %d”, 0, 1, 1,
                    “number of cols, [rows], [frames] in destination file\n\
                    (default = source size)”},
    {“-D”, “%d %d %d”, 0, 3, 2,
                    “number of cols, rows, [frames] in file dest file”},
    {“-filt”, “%s %s %s”, No, 3, 0, “\n\
                    filter name in x, y, and z (default = triangle)\n\
                    \”-filt ’?’\” prints a filter catalog”},
    {“-H”, “%~ %d %d %d”, 0, 4, 0,
            “\n\t[\”]row [column [frame]\”] specify pure image input size”,
            True},
    {“-s”, “%# %d %d %d %d”, 1, 5, 4,”source box (row col #rows #cols)”},
    {“-supp”, “%g %g %g”, -1, 3, 1, “filter support radius”},
    {“-xy”, “%b”, True, 1, 0, “filter x before y”},
    {“-yx”, “%b”, False,1, 0, “filter y before x”},
    {“-B”, “%N”, IFMT_BYTE, 1, 0, “input is BYTE formatted image”},
```

```
    {"-F", "%N", IFMT_FLOAT, 1, 0, "input is FLOAT formatted image"},
    {"-S", "%N", IFMT_SHORT, 1, 0, "input is SHORT formatted image"},
    {" [< ] in_file [> | ] out_file", NULL, 0, 0, 0, "end of information"},
    NULL};


main(int ac, char* av[])
{
double      xsupp, ysupp = -1., zsupp = -1.;
float       scale_factor;
Window_box      a, b;
int         cols, rows, frames, num_files
bool        color_out, dir_xy, img_format, no_headers, have_source
char        *xfiltname=FILTER_DEFAULT, *yfiltname=0, *zfiltname=0,
            **file_list;

            p_use_pound_sign(True);
            if ((num_files=parse_argus(&file_list, ac, av, arg_fmts,
                    &scale_factor,
                    &color_out,
                    &b.x, &b.x, &b.y, &b.z,
                    &b.x, &b.y, &b.z,
                    &xfiltname, &yfiltname, &zfiltname,
                    &no_header, &rows, &cols, &frames,
                    &have_source, &a.x0, &a.y0, &a.w, &a.h,
                    &xsupp, &ysupp, &zsupp,
                    &dir_xy, &dir_xy,
                    &img_format,
                    &img_format,
                    &img_format) < 0)
            exit(num_fils);

            while (num_files--){
                    in_fp = fopen(file_list[num_files], "r");
            ... ...
            }
exit        (0);
}
```

Now, we use a different arg_fmts[] to perform same functionality, so users can see how to manipulate argument formats:

```
arg_fmt_list_string    arg_fmts[] =    {
    {"-#", "%f", 1.0, 1, 1,
                "image will be shrinked or enlarged # times (-5.0 ~ +5.0)\n\
                # means '-real_number', such as: -1.1, --4.75, ..."},
    {"-c", "%+", True, 1, 0, "output color (RLE) image"},
    {"-d", "%d %d %d", 0, 3, 2,
```

```
                    “number of cols, rows, [frames] in file dest file”},
    {“-dc”, “%d”, 0, 1, 1,
                “number of cols in destination file. (default = input cols)”},
    {“-dr”, “%d”, 0, 1, 1,
                “number of rows in destination file. (default = input rows)”},
    {“-df”, “%d”, 0, 1, 1,
            “number of frames in destination file. (default = input frames)”},
    {“-filt”, “%s %s %s”, No, 3, 0, “\n\
                    filter name in x, y, and z (default = triangle)\n\
                    \”-filt ’?’\” prints a filter catalog”},
    {“-H”, “%~ %d %d %d”, 0, 4, 0,
            “\n\t[\”]row [column [frame]\”] specify pure image input size”},
    {“-s”, “%# %d %d %d %d”, 1, 5, 4,”source box (row col #rows #cols)”},
    {“-supp”, “%g %g %g”, -1, 3, 1, “filter support radius”},
    {“-xy”, “%1”, No, 1, 0, “filter x before y”},
    {“-yx”, “%0”, No, 1, 0, “filter y before x”},
    {“-B”, “%N”, IFMT_BYTE, 1, 0, “input is BYTE formatted image”},
    {“-F”, “%N”, IFMT_FLOAT, 1, 0, “input is FLOAT formatted image”},
    {“-S”, “%N”, IFMT_SHORT, 1, 0, “input is SHORT formatted image”},
    {“ [< ] in_file [> | ] out_file”, “0”, 0, 0, 0, “end of usage”},    NULL    };
```

The slight different in parse_argus() call is to change

```
                    &b.x, &b.x, &b.y, &b.z,
                    &b.x, &b.y, &b.z,
```

to

```
                    &b.x, &b.y, &b.z,
                    &b.x, &b.y, &b.z,
```

Program Level:

user

Related Functions:

arget(), avset(), parse_usage(), parserr()

Name:

parse_usage — display program usage and option formats

Synopsis:

parse_usage(arg_fmt_list_string* arg_fmt_list)

Arguments:

arg_fmt_list	argument format list string

Description:

parse_usage() displays program usage built by user argument format list.

Program Level:

kernel

Related Functions:

parse_argus()

Name:

parserr — * report parse_argus() errors

Synopsis:

```
parserr(    arg_fmt_lists   *fmt,
                int         n_val_less,
                int         cur_item,
                int         err_at_nth_argv,
                char        **argv          )
```

Arguments:

Description:

Program Level:

kernel

Related Functions:

parse_argus(), parse_usage()

Name:

peekbyte — ** peek a byte from a file stream

Synopsis:

```
peekbyte(    FILE    *fp,
             int     ch,
             int     job          )
```

Arguments:

fp	file stream pointer
ch	a character to be put back into pipe
job	SEEK_GETB, SEEK_UGETB, SEEK_PEEK

Description:

Program Level:

fuzzy

Related Functions:

Name:

pict_header_handle — * Mac PICT image header handle interface

Synopsis:

```
pict_header_handle(int        job,
                   U_IMAGE    *img,
                   ...        )
```

Arguments:

Description:

See (*img->header_handle)()

Program Level:

interface

Related Functions:

bridge_header_handle(), (*img->header_handle)()

Name:

pnm_header_handle — * PNM header handle

Synopsis:

```
pnm_header_handle(int       job,
                  U_IMAGE   *img,
                  ...       )
```

Arguments:

Description:

See (*img->header_handle)()

Program Level:

interface

Related Functions:

bridge_header_handle(), (*img->header_handle)()

Name:

pointer_buffer_size
p_buffer_size — get buffer size by passing buffer pointer

Synopsis:

pointer_buffer_size(VType *p)
p_buffer_size(VType *p)

Arguments:

p	buffer pointer

Description:

pointer_buffer_size() returns buffer size by given buffer pointer. It is a macro and same as *p_buffer_size*(), but much faster.

Program Level:

kernel

Related Functions:

nzalloc(), zalloc(), verify_buffer_size(), last_pointer_pos()

Name:

prgmerr — program error handler

Synopsis:

```
prgmerr(    int        err_type,
            char*      mesg_fmt,
            va_list    mesg_list    )
```

Arguments:

err_type	error type (level). Positive number is fatal level and causes program to terminate. 0 is the information level which report all error messages and return EOF. The negative level does same as level 0, but return ~level.
mesg_fmt	message and format. Same as C I/O format.
mesg_list	arguments for report.

Description:

prgmerr() is the standard way to handle the errors. The mesg_fmt and mesg_list are same as input and output format in C. Besides print these user message to stderr channel, *prgmerr*() also report the error status if variable errno is non zero (see error_mesg()).

Program Level:

user

Related Functions:

(*img->error)(), error_mesg(), message(), syserr()

Name:

pseudo_to_sep — * 8-bit color image to separated plane image converter

Synopsis:

```
pseudo_to_sep( byte      *out_r,
               byte      *in_8,
               int       size,
               cmap_t    *colormap[]    )
```

Arguments:

out_r	output buffer pointer at red channel
in_8	pseudo color data buffer pointer
size	image size (w*h)
colormap	pseudo color map

Description:

pseudo_to_sep() maps a pseudo color image into a separated plane color image. The size of input (in_8) image can be part of the image, but it must be equal to image width times number lines. If input buffer pointer (in_8) does not point to beginning of a line, then converted image will be shifted.

Program Level:

kernel

Related Functions:

Name:

pull_Itype — check the image type by looking the first few bytes in header

Synopsis:

pull_Itype(U_IMAGE *img)

Arguments:

img image structure pointer

Description:

Program Level:

kernel

Related Functions:

Name:

put_fits_head — write FITS header

Synopsis:

```
put_fits_head(  FITS_BASE*      fits_hd,
                U_IMAGE*        img,
                bool            completed    )
```

Arguments:

fits_hd	FITS header
img	image structure associated with fitd_hd
completed	end of FITS header. Since FITS can be written in many blocks, each of these block is 2880 bytes, so this flag is for the last block writing.

Description:

put_fits_head() write FITS header to a file according to image information stored in fits_hd structure, and returns number of lines write to img->OUT_FP. The FITS header line width is 80 characters long.

Program Level:

interface

Related Functions:

(img->header_handle)(HEADER_WRITE, img, ...)

Name:

put_superimpose_param — insert image superimpose information into HIPS header

Synopsis:

```
put_superimpose_param(   U_IMAGE        *img,
                         struct header  *hips_hd      )
```

Arguments:

img	image structure pointer
hips_hd	HIPS header into which the superimpose information in image structure will be inserted.

Description:

put_superimpose_param() puts image superimpose description into a HIPS header for generating a HIPS image with superimposed patterns.

Program Level:

assemble

Related Functions:

superimpose_add_elem(), superimpose_handle(), superimpose_images(), on_superimpose_elem()

Name:

quant_to_8 — quantizing 24-bit color image to 8-bit color image

Synopsis:

```
quant_to_8(     U_IMAGE  *img,
                int      num_color,
                cmap_t   *colormap[],
                VType    *buf          )
```

Arguments:

img	image structure pointer
num_color	number colors to be generated
colormap	color map to hold new color map elements
buf	output buffer to hold new data. The original data must be in img->src buffer.

Description:

Program Level:

kernel

Related Functions:

To_8()

Name:

ras8_to_rle — 8-bit Raster image to ILL (RLE) converter

Synopsis:

```
ras8_to_rle(    byte        *obp, *ibp,
                int         width,
                U_IMAGE     *img,
                cmap_t      *cmap[3],
                int         height          )
```

Arguments:

obp	output buffer pointer
ibp	input buffer pointer
width	ILL image line (scanline) width
img	image structure pointer
cmap	color map for 8-bit ILC image
height	number of rows to be converted

Description:

Program Level:

kernel

Related Functions:

Name:

rast_header_handle — ** Raster image header handle interface

Synopsis:

```
rast_header_handle(int        job,
                   U_IMAGE    *img,
                   ...        )
```

Arguments:

Description:

Program Level:

interface

Related Functions:

bridge_header_handle(), img->header_handle()

Name:

read_fits_image — * FITS image read interface

Synopsis:

```
read_fits_image(U_IMAGE*img,
                bool      file_type,
                bool      de_compress,
                bool      write_flag      )
```

Arguments:

img	image structure pointer
file_type	is in a FORTRAN format
de_compress	compressed data and need to be decompressed
write_flag	write decoded FITS data to img->OUT_FP

Description:

Program Level:

interface

Related Functions:

std_interface(), (*img->std_swif)()

Name:

read_hex_rgbmap — read hex RGB color map

Synopsis:

```
read_hex_rgbmap(U_IMAGE    *img,
                color_cell *cmbuf,
                int        mlen       )
```

Arguments:

img	image structure pointer
cmbuf	color map buffer
mlen	color map len for single channel

Description:

read_hex_rgbmap() reads hex number formatted RGBRGB color map and converts it into reg_cmap[] space.

Return:

1 in successful, or 0 in failure.

Program Level:

kernel

Related Functions:

Name:

read_pgm_image — * PGM image read interface

Synopsis:

```
read_pgm_image(U_IMAGE*img,
               int       max_value,
               int       format      )
```

Arguments:

img	image structure pointer
max_value	maximum pixel value in a PGM image file
format	PGM image format

Description:

Program Level:

interface

Related Functions:

std_interface(), (*img->std_swif)()

Name:

read_pict_image — * Mac PICT image read interface

Synopsis:

```
read_pict_image(U_IMAGE *img,
                bool        OsameI        )
```

Arguments:

img image structure pointer

OsameI do not change bit format, such 24-bit to 8-bit image converting

Description:

Program Level:

interface

Related Functions:

std_interface(), (*img->std_swif)()

Name:

read_pnm_image — * PNM images read interface

Synopsis:

```
read_pnm_image(U_IMAGE*img,
               int            max_value,
               int            format,
               xel**          xels,
          colorhash_table     cht,
               cmap_t         *colormap[]     )
```

Arguments:

img	image structure pointer
max_value	maximum pixel value in a PNM image file
format	PNM image format
xels	PNM data buffer (for reading)
cht	color hash table
colormap	regular color map

Description:

Program Level:

interface

Related Functions:

std_interface(), (*img->std_swif)()

Name:

read_ras_cmap — read regular color map

Synopsis:

```
read_ras_cmap(U_IMAGE*    img,
              int         mlen,
              int         chans        )
```

Arguments:

img	image structure pointer
mlen	color map length for single channel
chans	number of channels

Description:

read_ras_cmap() reads regular color map (RRR..., GGG..., BBB...) into CCS internal color map buffer - reg_cmap[3].

Return:

1 in successful, or 0 in failure.

Program Level:

kernel

Related Functions:

ReadRGBMap(), read_hex_rgbmap()

Name:

read_rast_image — * Raster image read interface

Synopsis:

```
read_rast_image(U_IMAGE*          img,
                struct rasterfile*  rast_hd,
                bool                O_same_I    )
```

Arguments:

img	image structure pointer
rast_hd	Raster image header structure pointer
O_same_I	do not change bit format

Description:

Program Level:

interface

Related Functions:

std_interface(), (*img->std_swif)()

Name:

read_rle_image — * RLE image read interface

Synopsis:

```
read_rle_image(U_IMAGE  *img,
               cmap_t   *local_map[],
               int      icon_factor    )
```

Arguments:

img	image structure pointer
local_map	regular color map to convert RLE color map
icon_factor	icon scale down factor

Description:

Program Level:

interface

Related Functions:

std_interface(), (*img->std_swif)()

Name:

read_sepplane_image — * separated plane image read interface

Synopsis:

```
read_sepplane_image(    U_IMAGE    *img,
                        int        skip,
                        int        frames      )
```

Arguments:

img	image structure pointer
skip	skip padding on each line
frames	number of frames to be read in

Description:

Program Level:

interface

Related Functions:

std_interface(), (*img->std_swif)()

Name:

read_tiff_image — * TiFF image read interface

Synopsis:

```
read_tiff_image(U_IMAGE*img,
                cmap_t    *colormap[],
                TIFF      *TiffIn,
                int       TiffRGB,
                bool      OsameI      )
```

Arguments:

img	image structure pointer
colormap	color map if applicable
TiffIn	TiFF header structure pointer
TiffRGB	
OsameI	do not change bit format

Description:

Program Level:

interface

Related Functions:

std_interface(), (*img->std_swif)()

Name:

read_var — * read variable length records from input file

Synopsis:

```
long
read_var(       byte*      ibuf,
                FILE*      fp,
                int        host_type,
                bool       isFORTRAN_FILE   )
```

Arguments:

ibuf	a buffer pointer for reading
fp	input file stream pointer
host_type	host type returned by check_host()
isFORTRAN_FILE	input file is a FORTRAN data file

Description:

read_var() is used for reading FORTRAN records on different endan machines.

Program Level:

kernel

Related Functions:

check_host(), int hostype, FTy;

Name:

readpipe — ** pipe reading handler

Synopsis:

```
readpipe(    byte    *buf,
             int     blocks, block_size,
             FILE    *fp                  )
```

Arguments:

buf	buffer for reading
blocks	number of blocks to read
block_size	block size to read
fp	file stream pointer

Description:

readpipe() is a soft tissue between user interface and kernel to overcome pipe seek issue on some OS. This routine is not suggested for user programming. Use init_piperead(img), (*img->read)(), (*img->seek)(), and (*img->write)() interfaces.

Program Level:

fuzzy

Related Functions:

img->read(), img->seek()

Name:

regmap_to_rlemap — regular color map to RLE colormap converter

Synopsis:

```
regmap_to_rlemap(    cmap_t     *colormap[],
                     int        nember_colors
                     int        channels,
                     rle_hdr    *rle_hd        )
```

Arguments:

colormap	regular color map
nember_colors	number colors in regular color map
channels	number channels in regular color map
rle_hdr	RLE header pointer

Description:

Program Level:

kernel

Related Functions:

rgbmap_to_othermap(), rlemap_to_regmap()

Name:

rgbmap_to_othermap — Raster color map to other map converter

Synopsis:

rgbmap_to_othermap(	color_cell*	colormap,
	int	nember_colors,
	int	to_reg_or_rle)

Arguments:

colormap	Raster color map
nember_colors	number of colors in Raster color maps
to_reg_or_rle	True to regular color map, and False to RLE color map

Description:

Program Level:

kernel

Related Functions:

regmap_to_rlemap(), rlemap_to_regmap()

Name:

rle_header_handle — * RLE header handle interface

Synopsis:

rle_header_handle(	int	job,
	U_IMAGE	*img,
	...	)

Arguments:

Description:

Program Level:

interface

Related Functions:

bridge_header_handle()

Name:

rlemap_to_regmap — RLE color map to regular color map converter

Synopsis:

```
rlemap_to_regmap(    cmap_t     *colormap[],
                     rle_hdr    *rle_hd        )
```

Arguments:

colormap	destination color map
rle_hdr	RLE header pointer

Description:

Program Level:

kernel

Related Functions:

regmap_to_rlemap(), rgbmap_to_othermap()

Name:

rotate90 — rotate an image in 90 degree

Synopsis:

```
rotate90(    VType    *in_buf, *o_buf,
             int      new_rows, new_cols,
             int      pixel_size,
             bool     rotate_left     )
```

Arguments:

in_buf	original image buffer
o_buf	rotated image buffer
new_rows, new_cols	new image dimensions
pixel_size	image pixel size
rotate_left	rotate counterclockwise

Description:

rotate images clockwise (default) by 90 degree

Program Level:

kernel

Related Functions:

color_rotate_90()

Name:

rtcp_getoptions — get RTP control header and options

Synopsis:

```
rtcp_getoptions(VType    *buf,
                int      num_options,
                ...                    )
```

Arguments:

buf	data buffer pointer
num_options	number of options to be get
..	option pairs (int option type, VType *option header)

Description:

rtcp_getoptions() try to get options in data buffer matching the option type and return the tail position of the RTP header (start data portion). If not option matching or num_options is zero, *rtcp_getoptions*() just skip header and options and return data starting position.

Example:

```
hdr_len = rtcp_getoptions(buffer, 2, RTPOPT_SDES, &rtcp_sdp,
                                     RTPOPT_FDES, &rtcp_fdp);
data = buffer + hdr_len;
```

Program Level:

kernel

Related Functions:

rtcp_setoption()

Name:

rtcp_setoption — set RTCP option header

Synopsis:

```
rtcp_setoption( rtpopthdr   *opt,
                char        *o_mesg,
                int         mesg_len,
                int         opt_type,
                bool        fin          )
```

Arguments:

opt	option header pointer
o_mesg	option data
mesg_len	option length in bytes
fin	final option

Description:

rtcp_setoption() copies RTP option header (o_mesg) into RTP header area (opt).

Return:

N/A

Program Level:

kernel

Related Functions:

rtcp_getoptions()

Name:

rtp_close — close real time protocol (RTP) socket

Synopsis:

```
void
rtp_close(   int          fd          )
```

Arguments:

fd	socket (file) descriptor

Description:

rtp_close() releases all RTP resources associated with *fd*, and close socket.

Return:

None

Program Level:

kernel

Related Functions:

rtp_open(), rtp_read(), rtp_write()

Name:

rtp_open — open real time protocol socket

Synopsis:

```
rtp_open(  char*   hostname,
           char*   port,
           int     carrier,
           int     mode,
           int     packet_size   )
```

Arguments:

hostname	either a host name or host IP address
port	either a port number or server name
carrier	SOCK_RAW, SOCK_DGRAM, or SOCK_STREAM; if nothing specified or specified wrong, then SOCK_DGRAM is used.
mode	True for transmitter, and False for receiver. ~mode is used for round trip (duplex). See Description for details.
packet_size	The frame will be broken downto number of packets, each with this size. The -1 packet_size for receiver means that kernel will not buffer incoming data at all, and every arriving packet will be directly dumped into user buffer regardless errors, but the returning messages are the same.

Description:

rtp_open() opens a socket and creates RTP resources for communication. The RTP can run over either network or transport protocol, such as IP or UDP, and in simplex or duplex mode. If mode is ~1, the *rtp_open* starts a transmitting duplex mode; if mode is ~0, then *rtp_open* starts a receiving duplex mode. The only difference between transmitting duplex and receiving duplex mode is how to initiate the RTP protocol.

The minimum packet size is 512 bytes, and the maximum packet size is 28KB.

Return:

-1 for critical errors. Zero for receiving a command packet (no data). A positive number represents how many bytes received, and a negative number represents how many bytes shorter than expected. For example:

rtp_read(socket_fd, buffer, 75)

requests to read at less 75% data of the entire frame. If entire frame size is one Megabytes, then RTP returns 911360 if it receives 890 KB data, or returns -51200 if it receives 700KB data.

If rtp_read() returns zero, then, rtp_receive_cmd() can be used to retrieve that command. Useful RTP communication commands are defined in rtp.h:

```
RTP_RETRANS    /* retrans certain area        */
RTP_REFILL     /* fill the hole */
```

Program Level:

kernel

Related Functions:

rtp_close(), rtp_read(), rtp_receive_cmd(), rtp_write()

Name:

rtp_read — read RTP frame

Synopsis:

```
rtp_read(   int        fd,
            void*      buf,
            int        tolerance     )
```

Arguments:

fd	a socket file descriptor
buf	buffer pointer for loading data
tolerance	at least this percentage of entire frame to be read

Description:

rtp_read() reads entire frame data without passing the length parameter. The tolerance parameter is used for computing the error rate. The tolerance range is 25 - 100.

There two mode for reading: 1) video and 2) image. If same data buffer (buf) is passed to *rtp_read*(), then *rtp_read*() considers this the video mode; otherwise, it is the image mode.

Return:

rtp_read() returns number bytes on successfully reading, negative number on less bytes reading, and -1 on failure. In video mode, the *rtp_read*() always returns number bytes read and do not give errors. If you do not need to use these alternative, use *rtp_read*() as regular read() system call except to pass the reading ratio (tolerance) instead of the reading size.

Program Level:

kernel

Related Functions:

rtp_close(), rtp_open(), rtp_write()

Name:

rtp_receive_cmd — retrieve RTP communication command

Synopsis:

```
rtp_receive_cmd(int          s,
        rtcp_movie_t    *rmp)
```

Arguments:

s	RTP socket ID
rmp	RTP movie header pointer; it can be NULL.

Description:

rtp_receive_cmd() is used to retrieve RTP communication command when *rtp_read*() returns zero. *rtp_receive_cmd*() returns RTP command, and rmp is used to pass further parameters for certain command unless you just want to see what command is sent.

Return:

command code defined in rtp.h

Program Level:

kernel

Related Functions:

rtp_read(), rtp_write()

Name:

rtp_recvbygroup — RTP group receiver

Synopsis:

```
rtp_recvbygroup(int      s,
                VType    *data,
                int      *datalen,
                int      *group_id,
                int      tolerance,
                bool     ignore_err     )
```

Arguments:

s	socket descriptor
data	data buffer pointer (returning)
datalen	data size received (returning)
group_id	data group ID pointer (in and returning)
tolerance	packets lost tolerance
ignore_err	send lost packets data to application anyway unless totally lost.

Description:

rtp_recvbygroup() uses RTP protocol in RTPCONT_MOVIE mode to receive data in frame group or other specified in content description for RTPCONT_TEST mode. The packets lost tolerance is a number from 1 - 89 to represent 1% - 89%. Any number out side this range is treated as 0% (no lost allowed). When the packet lost percentage is below tolerance or ignore_err is given, put data into data buffer and return data received in bytes; otherwise, wait for next group data. Total data length is returned by datalen pointer.

The group_id should be 0 when rtp_recvbygroup() first time being called. Once after returning or processing to next group data, the group_id will be set to the new group_id by rtp_recvbygroup().

Program Level:

kernel

Related Functions:

rtp_sendbygroup()

Name:

rtp_release_channel — RTP release channel resource

Synopsis:

rtp_release_channel(int channel)

Arguments:

channel	the channel ID of the resource to be release

Description:

rtp_release_channel() will free the resource allocated by rtp_recvbygroup() and set channel to be unused.

Program Level:

kernel

Related Functions:

rtp_request_channel(), rtp_recvbygroup()

Name:

rtp_request_channel — RTP request channel resource

Synopsis:

```
rtp_request_channel(    int     flow_id,
                        int     buffer_size,
                        int     protocol,
                        bool    retrans     )
```

Arguments:

flow_id	flow ID for known RTP channel, or zero for unknown RTP channel.
buffer_size	size of temporary buffer for restricted mode. User data will be buffered in kernel buffer and copied to user space when error is less than tolerance; otherwise, receiving data will be thrown away. If buffer size is zero, then, the receiving data is directly loaded into user space (user buffer), such as in video mode.
protocol	RTP protocol
retrans	retransmit when package lost

Description:

rtp_request_channel() try to find a unused channel, initialize it, and return it ID. It returns EOF if there is no channel available. The maximum channel resource is 64. This ID need to be passed in rtp_sendbygroup() as sender_id. If flow_id channel is been used by itself (flow_id == channel_index), *rtp_request_channel*() will overwrites the channel content; if flow_id channel is used by other flows, *rtp_request_channel*() will try to find a empty channel to use. This is more flexible, however, it is more complicated.

Program Level:

kernel

Related Functions:

rtp_release_channel(), rtp_sendbygroup()

Name:

rtp_sendbygroup — RTP group sender

Synopsis:

```
rtp_sendbygroup( int     s,
                 VType   *data,
                 int     data_len,
                 int     group_id,
                 int     sender_id,
                 char    *hdr,
                 int     hdr_len,
                 int     linewidth      )
```

Arguments:

s	socket descriptor
data	buffer pointer contains data to be sent
data_len	data size in bytes
group_id	data group ID
sender_id	flow control ID
hdr	users’ information or encoding method
hdr_len	users’ information size in byte
linewidth	image width (No for no care)

Description:

rtp_sendbygroup() breaks a large data chunk into small packets and send them through network by RTP protocol in RTPCONT_MOVIE or RTPCONT_TEST mode.

Program Level:

kernel

Related Functions:

rtp_recvbygroup()

Name:

rtp_setoption — set RTP transmission options

Synopsis:

```
rtp_setoption(int      s,
              int      job,
              int      ...      )
```

Arguments:

s	RTP socket descriptor
job	option ID defined in rtp.h
...	option parameters

Description:

rtp_setoption() is used to set up RTP transmitting options, such as packet delay, quality of service, etc. Available options are:

RTP_OPT_UDELAY	packet delay in micro seconds
RTP_OPT_HDRLEN	set header length
RTP_OPT_GROUPID	change group ID
RTP_OPT_ADDFLAG	add flags (|); available flags are:

RTP_RETRANS	need to retransmit
RTP_REFILL	need to refill a certain area
RTP_LOCALBUF	read into local buffer (default)
RTP_NOINPOLA	don't interpolate missing regions
RTP_SLOWVIDEO	in this manner, kernel will buffer the incoming packet, look at the position in header, and move the data to the right place in user buffer space. It is about 30% slower on Sparc stations, but may have better service quality. Is that right?

RTP_OPT_FLAGS	change flags
RTP_OPT_QOS	set quality of service

Return:

0 on successful; otherwise for failure.

Program Level:

kernel

Related Functions:

rtp_close(), rtp_open(), rtp_read(), rtp_write()

Name:

rtp_write — write RTP frame

Synopsis:

```
rtp_write(  int       fd,
            void*     buf,
            int       length      )
```

Arguments:

fd	socket file descriptor
buf	buffer contains data to be written
length	number of bytes data to be written

Description:

rtp_write() writes data to a RTP socket opened by *rtp_open*(). *rtp_write*() returns number of bytes written or negative value for failure.

Return:

number of bytes written on successful; negative value for failure.

Program Level:

kernel

Related Functions:

rtp_close(), rtp_open(), rtp_read()

Name:

s_buf_size — * change socket buffer

Synopsis:

s_buf_size(	int	socket,	
	int	buf_size	)

Arguments:

socket	socket descriptor created by build_socket() or socket()
buf_size	new buffer size

Description:

s_buf_size() changes socket default buffer size

Program Level:

kernel

Related Functions:

build_socket()

Name:

select_color_form — * select right color format for type conversion

Synopsis:

```
select_color_form(      U_IMAGE     *img,
                        bool        out_same_as_input       )
```

Arguments:

img	image structure pointer
out_same_as_input	do not change 24-bit color format to 8-bit color when set up color conversion flags

Description:

select_color_form() is used after kernel reads an image header for set up conversion flags according to *format_init*() or other setups by user program.

Program Level:

kernel

Related Functions:

bridge_header_handle()

Name:

set_pr_colormap — ** map regular color map to a Raster color map

Synopsis:

```
colormap_t*
set_pr_colormap(cmap_t*   rg_map[3],
                int       num_colors     )
```

Arguments:

rg_map	regular color map
num_colors	number of colors in regular color map

Description:

set_pr_colormap() creates a Raster color map and map regular color map to it content.

Program Level:

fuzzy

Related Functions:

set_gs_colormap()

Name:

set_gs_colormap — ** set a linear Raster color map

Synopsis:

```
colormap_t*
set_gs_colormap(void          )
```

Arguments:

None

Description:

set_gs_colormap() creates a Raster color map and initial it to a linear map.
c[0]= 0, c[1] = 1, ...c[n] = n

Program Level:

fuzzy

Related Functions:

set_colormap()

Name:

set_time_limit — * set time limit for license time out

Synopsis:

set_time_limit(int year, month, day, hour, min)

Arguments:

date and time

Description:

set time limit for if_time_exp() and exit_timeout() routines.

Program Level:

user

Related Functions:

exit_timeout(), if_time_exp()

Name:

snf_to_rle — convert ILC (Raster) image to ILL (RLE) image

Synopsis:

```
snf_to_rle(    byte    *obp, *ibp,
               byte    *colormap[]    )
```

Arguments:

obp	output buffer pointer
ibp	input buffer pointer
colormap	if input is an 8-bit color image, the color map need be passed in here.

Description:

snf_to_rle() converts interleave cell (ILC, Raster) image to interleave line (ILL, RLE) image format

Program Level:

kernel

Related Functions:

any_color_to_rle(), any_to_seplane(), ilc_transfer(), line_to_cell_color(), line_to_sep_color(), map8_gray(), ras8_to_rle()

Name:

socket_connect — make connection after built a socket

Synopsis:

```
socket_connect(int                 sfd,
         struct sockaddr_in*  s_addr,
                int           trans,
                int           port,
                int           options,
                int           *ov, ol       )
```

Arguments:

sfd	socket descriptor
s_addr	socket address structure pointer
trans	True for transmitter, False for receiver
port	TCP_SOCK, RTP_SOCK, or UDP_SOCK
options	options for setsockopt()
ov	pointer of option data
ol	option data size

Description:

socket_connect() try to connect two host through a socket. In TCP socket, it will wait until connection is built. For RTP socket, it returns a window size agreed by both hosts. For udp and IP, it will do nothing.

Program Level:

kernel

Related Functions:

build_socket()

Name:

std_interface — * standard interface for image data handling

Synopsis:

```
std_interface(  int       job,
                U_IMAGE   *img,
                ...       )
```

Arguments:

job	job description: FI_LOAD_FILE FI_LOAD_ROW FI_RLOAD_BUF FI_RESERVED FI_ACCESS_ABS_FRAME FI_SAVE_FILE FI_WHAT_FILE FI_DESC_ETA FI_GET_INFORMATION FI_HIPS_HEADER_FORMAT FI_INIT_NAME FI_PNM_MAXVAL
img	image structure pointer
...	arguments. Details are in Description

Description:

```
int     (*img->std_swif)(FI_LOAD_FILE, img,
                        char* name, bool multi_frames / OsameI)

int     (*img->std_swif)(FI_RLOAD_BUF, img,
                        char* buf, bool multi_frames / OsameI)

int     (*img->std_swif)(FI_ACCESS_ABS_FRAME, img,
                        char* buf, int nth_frame);
```

All these three function return loading data size if successful, or EOF on failure. The FI_LOAD_FILE will check the buffer size (img->src). If it is not correct std_swif() will change the old buffer to a new one. For FI_RLOAD_BUF and FI_ACCESS_ABS_FRAME, the buf argument must be allocated and passed to std_swif().

name	the name of caller or any information that you want to pass to the function for tracing errors.
multi_frame	(for RLE image or other multi-frame multi-header images) tells loading routine not to rewind or advance this file for any purpose since the rest frames will be loaded in order. Other multi-frame images, the control variable ‘img->load_all’ controls loading process to load next N (= img->load_all) frames. N=0 & N=1 are same.

The OsameI tells loading routine not to change the image format, such as 8-bit color to 24-bit, or 24-bit color to 8-bit transform.

nth_frame load nth frame data for FI_ACCESS_ABS_FRAME function.

bool (*img->std_swif)(FI_SAVE_FILE, img,
char* buffer/bool map, int nth_frame/LKT* lkt); /* for HIPS | RLE */

bool (*img->std_swif)(FI_SAVE_FILE, img,
char* buffer/bool map, int alpha / reverse); /* for Sun_Raster | TiFF */

If no map or buffer is NULL, save img->src. Otherwise, save img->dest or buffer. For saving multiple frame image, the control variable 'img->save_all' functions as same as the 'img->load_all' in saving control.

nth_frame save nth frame in img->src for HIPS images

lkt for mapping RLE image

alpha When output is Sun-Raster file, alpha flag generates alpha channel.

reverse When Output is TiFF image, reverse flag will swap RGB to BGR or another way around.

bool (*img->std_swif)(FI_DESC_ETA, img, char* info, CER* cer)
Add Elastic Tuning information into header description field

bool (*img->std_swif)(FI_INIT_NAME, img, char* name, 0);
initial header with name string.

All above bool functions return 0 on successful, or -1 on failure.

xel** (*img->std_swif)(FI_PNM_MAXVAL, img, &max_value);
It returns PNM data buffer pointer and the maximum pixel value in max_value.

Program Level:
interface

Related Functions:
bridge_header_handle(), (*img->std_swif)(), format_init()

Name:

syserr — system error handle

Synopsis:

syserr(	char	*format,	
	va_list	messages	)

Arguments:

format	information format string
messages	user's messages and values

Description:

syserr() report error message and user's message and exit the program.

Program Level:

kernel

Related Functions:

error_mesg(), message(), prgmerr()

Name:

tiff_header_handle — * TiFF image header handle interface

Synopsis:

```
tiff_header_handle(int       job,
                   U_IMAGE   *img,
                   ...       )
```

Arguments:

Description:

Program Level:

interface

Related Functions:

bridge_header_handle(), (*img->header_handle)()

Name:

tvadd, tvsub— time value addition and substraction

Synopsis:

```
void
tvadd(          timeval    *tv1,
                           *tv2          )

void
tvsub(          timeval    *tv1,
                           *tv2          )
```

Arguments:

tv1	first time and result
tv2	second time value

Description:

tvadd() calculates tv1 = tv1 + tv2
tvsub() calculates tv1 = tv2 - tv1

Program Level:

assemble

Related Functions:

tvdiff()

Name:

tvdiff — difference of time value

Synopsis:

```
float
tvdiff(        timeval     *tv1,
                            *tv2        )
```

Arguments:

tv1	first time value
tv2	second time value

Description:

returns tv2 - tv1without changing any of these tvs contents.

Program Level:

assemble

Related Functions:

tvadd(), tvsub()

Name:

ungetbyte — **

Synopsis:

```
ungetbyte(    int      ch,
              FILE     *fp          )
```

Arguments:

ch	character to be put back file stream
fp	file stream pointer

Description:

Program Level:

fuzzy - soft tissue

Related Functions:

Name:

usage_n_options — print usage options

Synopsis:

```
usage_n_options(    char*     message,
                    int       nth_argv,
                    char*     nth_option    )
```

Arguments:

message	message string
nth_argv	nth argument is wrong
nth_option	the wrong option information

Description:

Program Level:

kernel

Related Functions:

prgmerr(), syserr()

Name:

verify_buffer_size — get the correct size buffer

Synopsis:

```
verify_buffer_size(    VType    **pp,
                       int      blocks_N_type,
                       int      bytes_in_block,
                       char     *caller_name    )
```

Arguments:

pp — buffer or pointer address

blocks_N_type — number of blocks and its type. A positive block number to get new buffer, and a negative block number to get a reallocated buffer, if and only if the new blocks * bytes_in_block is greater then the current buffer size.

bytes_in_block — block size

caller_name — the calling routine name or any reference name.

Description:

verify_buffer_size() checks the given buffer size with new size [abs(blocks) * bytes_in_block]. If the buffer size smaller then the new size, it will append a piece of memory to old buffer when number blocks is less then 0, or reallocate a buffer when the number of blocks is greater than 0 and free the old buffer.

Return:

verify_buffer_size() returns address when a new buffer has been allocated, or NULL if nothing is changed.

In case of memory allocation failed in allocating new mode (blocks_N_type is positive), *verify_buffer_size()* prints error message, and exits from system if the caller name is given, or return NULL if the caller name is NULL or No. In failure of reallocating mode, *verify_buffer_size()* only returns NULL.

Program Level:

kernel

Related Functions:

nzalloc(), zalloc(), pointer_buffer_size(), last_pointer_pos()

Name:

vff_def_cmap — generate default VFF color map

Synopsis:

vff_def_cmap(U_IMAGE *img)

Arguments:

Description:

Return:

Program Level:

kernel

Related Functions:

Name:

vfft
vfft2d
vfft3dre
vfft_2d
vfftn
vrft2d
vrft3d
vrft_2d — virtual Fast Fourier Transform

Synopsis:

```
vfft(         float     *re_plane,
              float     *im_plane,
              int       points_in_log2  )

vfft2d(       float     *re_plane,
              float     *im_plane,
              int       points_in_log2  )

vfft3d(       VType     *src,
              int       x, y, z,
              COMPLEX   *dst            )

vfft_2d(float *re_plane,
              float     *im_plane,
              int       rows_in_log2,
              int       cols_in_log2    )

vfftn(        float     *re_plane,
              float     *im_plane,
              int       points_in_log2  )

vrft2d(       VType     *src,
              unsigned  x, y,
              VType     *dst            )

vrft3d(       COMPLEX   *src,
              int       x, y, z,
              FType     *dst            )

vrft_2d(      float     *re_plane,
              float     *im_plane,
              int       rows_in_log2,
              int       cols_in_log2    )
```

Arguments:

Description:

Program Level:

kernel

Related Functions:

Name:

w_init
w_load — *

Synopsis:

w_init(	MType	half_len	)
w_load(	int	points	)

Arguments:

Description:

Program Level:

kernel

Related Functions:

Name:

write_rast — write image to a Sun Raster image file

Synopsis:

```
write_rast(     U_IMAGE  *img,
                bool     alpah_channel,
                int      RAST_TYPE   )
```

Arguments:

img	image structure pointer
alpah_channel	add alpha channel in lumin mode
RAST_TYPE	Sun Raster format

Description:

Program Level:

kernel

Related Functions:

Name:

write_rle — write image to a RLE image file

Synopsis:

```
write_rle(     U_IMAGE  *img,
               LKT      *lkt,
               bool     mapped        )
```

Arguments:

img	image structure pointer
lkt	lookup table for re-mapping colors
mapped	use image in img->dest buffer; otherwise use img->src buffer.

Description:

Program Level:

kernel

Related Functions:

cmap_t *reg_cmap[3]

Name:

x_extender — extended X server

Synopsis:

x_extender(bool udp)

Arguments:

udp	UDP server or TCP server (False)

Description:

x_extender() checks if any data come in from network. If does, assemble it and send it to application with returning True; otherwise return False. If data come with command options (e.g. from sendto program), then you can retrieve them by calling *get_args()* sub-routine.

Program Level:

kernel

Related Functions:

x_extender_init(), build_arg_list(), get_args()

Name:

x_extender_init — X extended server initial routine

Synopsis:

x_extender_init(	char*	server_name	
	bool	udp,	
	int	buf_size	)

Arguments:

server_name	server name or port number in string format
udp	UDP server of TCP (False)
buf_size	window size for packet transmission

Description:

x_extender_init() starts a TCP bases server for x_extender() routine to receive data from network.

Program Level:

kernel

Related Functions:

x_extender()

Name:

zalloc — allocate memory with all cells cleaned to zero

Synopsis:

```
VType*
zalloc(    int      blocks,
           int      size,
           char     *caller_name    )
```

Arguments:

blocks	number of blocks
size	size of each block.
caller_name	name for error trace. It can be procedure name, or any name to be reference.

Description:

zalloc() allocates memory space with size = *blocks* * *size*. Also, it cleans all of space to zero. All memory handle functions and subroutines are tied to kernel debug, so once the global variable *"debug"* is turned on (any level), then debug will print all memory information whenever memory allocating status are changed.

In case of memory allocation failed, *zalloc()* prints error message, and exits from system if the caller name is given, or return NULL if the caller name is NULL or No.

ZALLOC() is the macro for *zalloc*() and less overhead.

Program Level:

kernel

Related Functions:

nzalloc(), pointer_buffer_size(), verify_buffer_size(), last_pointer_pos()

Name:

—

Synopsis:

Arguments:

Description:

Return:

Program Level:

kernel

Related Functions:

INDEX

W

X

Z