University of Amsterdam
Faculty of Mathematics,
Computer Science, Physics
and Astronomy

Kruislaan 403
1098 SJ Amsterdam,
The Netherlands

TNO Institute of
Applied Physics

P.O. Box 155
2600 AD Delft, The Netherlands
Stieltjesweg 1
2628 CK Delft, The Netherlands

# SCIL_Image

version 1.4

# Reference Manual

May, 1998

# Reference Manual Pages

## abort

***NAME***

abort - abort program abnormally

***SYNOPSIS***

```
void abort(void)
```

***DESCRIPTION***

This function is an interface to the standard C function as implemented on the current system. The functionality of this function is:

abort() causes the program to terminate abnormally, as if by raise(SIGABRT).

***RETURN VALUES***

The function does not return;

***SEE ALSO***

exit  _exit

## *abs*

## *labs*

***NAME***

abs, labs - integer absolute value

***SYNOPSIS***

```
int abs(int i)

long labs(long n)
```

***DESCRIPTION***

These functions are interfaces to the standard C functions as implemented on the current system. The functionality of these function are:

abs() returns the absolute value of its integer operand.

labs() returns the absolute value of its long operand

***BUGS***

You get what the hardware gives on the smallest integer.

***SEE ALSO***

floor  fabs

## *abs_im*

***NAME***

abs_im - absolute value or modulus of an image

***SYNOPSIS***

```
#include "im_proto.h"

int abs_im(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

Calculate the absolute value of each element of image "in" and store the result in the corresponding element of image "out". If the image "in" is a complex image the modulus will be calculated. In this case if image "out" is also an complex image then the result will be stored in the real part of each element of "out" and the imaginary part will be cleared. If "out" is not a complex image then the result will be a float image.

***NOTE***

For more powerful image arithmetic expressions (scaling, adding offsets, etc.), use the function eval().

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

absd_im eval

## ***absd_im***

***NAME***

absd_im - absolute difference of images

***SYNOPSIS***

```
#include "im_proto.h"

int absd_im(IMAGE *in1, IMAGE *in2, IMAGE *out)
```

***DESCRIPTION***

Calculate the absolute value of the difference between each element of image "in1" and the corresponding element of image "in2" and store the result in the corresponding element of image "out".

***NOTE***

For more powerful image arithmetic expressions (scaling, adding offsets, etc.), use the function eval().

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

abs_im  eval

## *add_applic_exposure_func*

***NAME***

add_applic_exposure_func - add function to exposure list

***PLATFORM***

UNIX, Macintosh.

***SYNOPSIS***

```
#include "disp_p.h"

int add_applic_exposure_func(void (*func)(IMAGE *ip))
```

***DESCRIPTION***

This function adds the specified function "func" to the list of COMPILED functions that are to be called when an exposure event occurs. The function that will be called must have one parameter only, an IMAGE pointer:

```
void user_exposure(IMAGE *image)
```

The name "user_exposure" is an example only, any name may be used for such a function.

***NOTE***

A new and better interface for application programs to show their interest in events has become available by means of the functions im_exposure_func() and im_input_func(). The support of the functions applic_exposure(), add_applic_exposure_func(), applic_win_input() and add_applic_win_input_func() is no longer guaranteed in future versions of SCIL_Image.

***RETURN VALUES***

None

***SEE ALSO***

applic_exposure  applic_win_input  add_applic_win_input_func im_exposure_func im_input_func

## *add_applic_win_input_func*

***NAME***

add_applic_win_input_func - add function to window input list

***PLATFORM***

UNIX, Macintosh.

***SYNOPSIS***

```
#include "disp_p.h"

int add_applic_win_input_func(void (*func)(IMAGE *, int, int, char,
int))
```

***DESCRIPTION***

This function adds the specified function "func" to the list of COMPILED functions that are to be called when a window input event occurs. The function that will be called must have the following parameters:

```
void user_win_input(IMAGE *image, int xpos, int ypos, char ch,
int but)
```

"image" is the pointer to the image in which the event occurred. "xpos" and "ypos" are the position of the pointer in the image (not in the window). "ch" is the character of the key that is pressed. "but" is the button that of the mouse that is pressed or released

The name "user_win_input" is an example only, any name may be used for such a function.

***NOTE***

A new and better interface for application programs to show their interest in events has become available by means of the functions im_exposure_func() and im_input_func(). The support of the functions applic_exposure(), add_applic_exposure_func(), applic_win_input() and add_applic_win_input_func() is no longer guaranteed in future versions of SCIL_Image.

***RETURN VALUES***

None

***SEE ALSO***

applic_win_input applic_exposure add_applic_exposure_func point_im EventType IsMouseDown KeyPressed MouseMove MousePress MouseRelease

## *add_complex*

## *sub_complex*

## *mul_complex*

## *div_complex*

***NAME***

add_complex, sub_complex, mul_complex, div_complex - complex arithmetic

***SYNOPSIS***

```
#include "im_proto.h"

int add_complex(IMAGE *in, double real_part, double imaginary_part,
IMAGE *out)

int sub_complex(IMAGE *in, double real_part, double imaginary_part,
IMAGE *out)

int mul_complex(IMAGE *in, double real_part, double imaginary_part,
IMAGE *out)

int div_complex(IMAGE *in, double real_part, double imaginary_part,
IMAGE *out)
```

***DESCRIPTION***

Complex arithmetic. The operation is performed on image "in" and the result is stored in image "out". The complex value is specified by "real_part" and "imaginary_part".

add_complex() adds the complex value to each element of "in".

sub_complex() subtracts the complex value from each element of "in".

mul_complex() multiplies each element of "in" with the complex value.

div_complex() divides each element of "in" by the complex value. If divisions by zero occur, an error will be generated.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

add_int  sub_int  mul_int  div_int  add_float  mul_float  sub_float  div_float

## *add_float*

## *sub_float*

## *mul_float*

## *div_float*

***NAME***

add_float, sub_float, mul_float, div_float - floating point arithmetic

***SYNOPSIS***

```
#include "im_proto.h"

int add_float(IMAGE *in, double constant, IMAGE *out)

int sub_float(IMAGE *in, double constant, IMAGE *out)

int mul_float(IMAGE *in, double constant, IMAGE *out)

int div_float(IMAGE *in, double constant, IMAGE *out)
```

***DESCRIPTION***

Floating point arithmetic. The operation is performed on image "in" and the result is stored in image "out".

add_float() adds "constant" to each element of "in".

sub_float() subtracts "constant" from each element of "in".

mul_float() multiplies each element of "in" with "constant".

div_float() divides each element of "in" by "constant". If divisions by zero occur, an error will be generated.

These functions have the extension "_float" because C does not allow different types to be passed through the same parameter. The name "_float" has been used to indicate a floating point value.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

add_int sub_int mul_int div_int add_complex sub_complex mul_complex div_complex

## *add_im*

## *sub_im*

## *mul_im*

## *div_im*

***NAME***

add_im, sub_im, mul_im, div_im - image arithmetic

***SYNOPSIS***

```
#include "im_proto.h"

int add_im(IMAGE *in1, IMAGE *in2, IMAGE *out)

int sub_im(IMAGE *in1, IMAGE *in2, IMAGE *out)

int mul_im(IMAGE *in1, IMAGE *in2, IMAGE *out)

int div_im(IMAGE *in1, IMAGE *in2, IMAGE *out)
```

***DESCRIPTION***

add_im() adds each element of image "in1" to the corresponding element of image "in2" and stores the result in image "out"

sub_im() subtracts each element of "in2" from the corresponding element of "in1" and stores the result in "out"

mul_im() multiplies each element of "in1" with the corresponding element of "in2" and stores the result in "out"

div_im() divides each element of image "in1" by the corresponding element of image "in2" and stores the result in image "out". In the event that a pixel of "in2" is equal to zero, the corresponding pixel in "out" will be set to the value of the corresponding pixel of "in1".

***NOTE***

For more powerful image arithmetic expressions (scaling, adding offsets, etc.), use the function eval() .

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)
When division by zero occur, div_im() returns an error-count (number of divisions by zero)

***SEE ALSO***

eval

## *add_int*

## *sub_int*

## *mul_int*

## *div_int*

***NAME***

add_int, sub_int, mul_int, div_int - integer arithmetic

***SYNOPSIS***

```
#include "im_proto.h"

int add_int(IMAGE *in, int constant, IMAGE *out)

int sub_int(IMAGE *in, int constant, IMAGE *out)

int mul_int(IMAGE *in, int constant, IMAGE *out)

int div_int(IMAGE *in, int constant, IMAGE *out)
```

***DESCRIPTION***

Integer arithmetic. The operation is performed on image "in" and the result is stored in image "out".

add_int() adds "constant" to each element of "in".

sub_int()subtracts "constant" from each element of "in".

mul_int()multiplies each element of "in" with "constant".

div_int divides each element of "in" by "constant". If divisions by zero occur, an error will be generated.

These functions have the extension "_int" because C does not allow different types to be passed through the same parameter. The name "_int" has been used to indicate an integer value.

***NOTE***

For more powerful image arithmetic expressions (scaling, adding offsets, etc.), use the function eval() .

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

add_im  sub_im  mul_im  div_im  eval

## *AddImageInfo*

## *GetImageInfo*

## *RemoveImageInfo*

***NAME***

AddImageInfo - add auxiliary information to an image

GetImageInfo - retrieve pointer to auxiliary information of an image

RemoveImageInfo - remove auxiliary information from an image

***SYNOPSIS***

```
#include "im_infra.h"

int AddImageInfo(IMAGE *im, char *name, void *info, void
(*dfunc)(void *))

void *GetImageInfo(IMAGE *im, char *name)

int RemoveImageInfo(IMAGE *im, char *name)
```

***DESCRIPTION***

The Image Info mechanism allows for storage of auxiliary information with images. The information is stored using a user supplied name that is used as its identification from that moment on. The pointer to the information in combination with the identification string is stored in a list that is attached to the IMAGE structure. At any given time this pointer can be retrieved and the data viewed or processed.

AddImageInfo() stores the pointer "info" with image "im" using the string "name" as its identification. The function pointer "dfunc" can be used to automatically destroy the information when the image is destroyed, NULL meaning no automatic destruction. This function must have only one parameter, being a pointer to the information.

Please note that only a pointer to the data is stored and NOT the data itself, so be sure that the data remains accessible as long as the pointer is kept with the image. This means that you can only use global data or dynamically allocated memory.

GetImageInfo() retrieves the pointer to the information "name" that is stored with image "im".

RemoveImageInfo() removes the information "name" from image "im". If a destruction function was specified for this information, it is called with the pointer to the information as its argument.

The Image infrastructure has no knowledge of the contents of the information, meaning that the application (programmer) remains responsible for its integrity. The mechanism only provides the service of keeping a pointer to the information with the image(s).

***RETURN VALUES***

| | |
|---|---|
| AddImageInfo: | IE_OK (1) on success or<br>IE_NOT_OK (0) when already present or no more memory available. |
| GetImageInfo: | a pointer to the information or NULL when not present |
| RemoveImageInfo: | IE_OK (1) on success or<br>IE_NOT_OK (0) on failure to remove. |

## *aim_readfile*

***NAME***

aim_readfile - read an image from a file in AIM format

***SYNOPSIS***

```
#include "im_proto.h"

IMAGE *aim_readfile(char *filename, IMAGE *image, int xpos, int ypos)
```

***DESCRIPTION***

Read the image stored in the AIM format file "filename" and put it in image "image". If "USE_NAME" (a NULL pointer) is specified as the image, a new image is created at position "xpos", "ypos", with the same name as the file. If an image is already present with that name, that image will be used.

The data-files of the AIM format must have the ".im" extension. Data-files for which no header file with the extension ".hd" is present, are assumed to contain a 256 * 256 grey value image.

"filename" may be specified with or without the mandatory extension ".im", it is appended when necessary.

***RETURN VALUES***

The pointer to the image in which the data was put, either an existing image or a newly created one.
NULL on failure

***SEE ALSO***

readfile  ics_readfile  tiff_readfile  tcl_readfile  writefile

## *aio_label*

***NAME***

aio_label - image labeling without building an object list

***SYNOPSIS***

```
#include "im_aio.h"

int aio_label(IMAGE *in, IMAGE *out, int connect)
```

***DESCRIPTION***

aio_label() labels the binary objects in the binary image "in" and puts the result in the labeled image "out". The connectivity "con" can either be 4 or 8.

The recursive labeling algorithm tries not to use the same label twice on a horizontal scanline.

***RETURN VALUES***

The number of objects labeled on success.
Negative error status on failure (see im_error.h).

***SEE ALSO***

list_label

## *all_im*

***NAME***

all_im - perform an operation on all or a selection of images

***SYNOPSIS***

```
#include "im2scil.h"

void all_im(char *command, int type)
```

***DESCRIPTION***

The operation specified by "command" is performed on all existing images that are of type "type". If the type of the image is not relevant, "ALL" (=0) can be specified to perform the operation on all images. In the parameter list of the operation the image name must be replaced by the character "$" (see below). All occurrences of the "$" character in the argument list of an operation will be replaced by the same image name. This means that operations will be performed in place when specifying both input and output image with the "$".

***EXAMPLE***

clearing all images:

```
all_im("clear_im $",ALL);
```

in place thresholding of all grey 2d images with threshold value 100:

```
all_im("thresh $ $ 100",GREY_2D);
```

***RETURN VALUES***

None

## *anchor_skelet*

***NAME***

anchor_skelet - anchor skeleton

***SYNOPSIS***

```
#include "im_proto.h"

int anchor_skelet(IMAGE *in, IMAGE *mask, IMAGE *out, int iter, int
endp, int bound)
```

***DESCRIPTION***

Change the objects in the binary image "in" into anchor skeletons and store the result into the binary image "out".

The anchor skeleton is a special variant of skeletonization (see also "hild_skelet" for "normal" skeletonization). During each thinning cycle the so-called anchor points are forced to be skeleton pixels, even if they do not meet the conditions. These anchor points are the object pixels of the mask image "mask". The skeleton is "anchored" through these points. If the anchor belongs to an object (or is connected with it) the connectivity of the original skeleton points and the anchor points is maintained. In this case the resulting skeleton will deviate from the medial axis of the object, in the direction of the anchor points. The result of an anchor skeletonization is not readily described. The command is recommended primarily for special applications, applied by rather experienced users.

The thinning operation may be executed for only a limited number of cycles, as specified by the parameter "iter". Full skeletonization results if the value 0 is specified for this parameter. The operation then continues until no more pixels are deleted.

"endp" specifies that the endpixels of the skeleton must be preserved with each thinning iteration (1 is preserve, 0 is do not preserve).

"bound" specifies that the edge around the image must be set to foreground (1) or to background (0) pixels.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

holt_skelet  hild_skelet  skelpoints

## *and_im*

***NAME***

and_im - bitwise and of images

***SYNOPSIS***

```
#include "im_proto.h"

int and_im(IMAGE *in1, IMAGE *in2, IMAGE *out)
```

***DESCRIPTION***

Perform a bitwise AND operation of each element of "in1" with the corresponding element of "in2" and store the result in "out"

***NOTE***

For more powerful image arithmetic expressions (scaling, adding offsets, etc.), use the function eval().

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

or_im  xor_im  invert_im  shift_im

---

## *angle*

***NAME***

angle - obtain angle of object

***SYNOPSIS***

```
#include "im_aio.h"

double angle(LIST *link)
```

***DESCRIPTION***

AIO primitive to obtain value of an object feature

angle() returns the angle of the object pointed to by "link", if this feature has previously been measured with either measure() or object_shape_meas(). The angle is measured in radians relative to the X-axis

***RETURN VALUES***

angle of object in degrees on success.
0.0 if link is not an object or if angle has not been measured.

***SEE ALSO***

measure  object_shape_meas  object_dens_meas

---

## *angle_detection*

***NAME***

angle_detection - line angle detector

***SYNOPSIS***

```
#include "im_proto.h"

int angle_detection(IMAGE *in, IMAGE *out, double thres)
```

***DESCRIPTION***

Detection of angles in skeleton segments. The binary image "in" is scanned for individual skeleton segments. For each skeleton segment the end-points (say A and B) are detected and a straight line AB is drawn. Then the point P on the skeleton segment is detected with maximum distance to the line element AB and the angle between the line elements AP and BP is calculated. If this angle exceeds a certain threshold value, as specified by the parameter "thres" the point P is detected as an angle point and becomes an object pixel in the binary image "out".

***NOTE***

This command is only meaningful if the image "in" is a skeleton image.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

## *applic_exposure*

***NAME***

applic_exposure - ask to be notified on an exposure event

***PLATFORM***

UNIX, Macintosh.

***SYNOPSIS***

```
#include "disp_p.h"

int applic_exposure(int send_events, int skip_when_busy)
```

***DESCRIPTION***

With this function an application can indicate that it is interested in exposure events of image-windows. The user can specify a (list of) COMPILED function(s) that are to be called whenever an exposure event comes along. A COMPILED function can be added to the list with add_applic_exposure_func(). If there is no list of COMPILED functions an INTERPRETED function "handle_exposure()" will be called. "skip_when_busy" determines whether handle_exposure() can be called again when it has not finished the previous event yet.

***NOTE***

A new and better interface for application programs to show their interest in events has become available by means of the functions im_exposure_func() and im_input_func(). The support of the functions applic_exposure(), add_applic_exposure_func(), applic_win_input() and add_applic_win_input_func() is no longer guaranteed in future versions of SCIL_Image.

***RETURN VALUES***

None

***SEE ALSO***

add_applic_exposure_func  applic_win_input add_applic_win_input_func

## *applic_win_input*

***NAME***

applic_win_input - ask to be notified on an window input event

***PLATFORM***

UNIX, Macintosh.

***SYNOPSIS***

```
#include "disp_p.h"

int applic_win_input(int send_events, int skip_when_busy)
```

***DESCRIPTION***

With this function an application can indicate that it is interested in input events in windows. The user can specify a (list of) COMPILED function(s) that are to be called whenever an input event comes along. A COMPILED function can be added to the list with add_applic_win_input_func(). If there is no list of COMPILED functions an INTERPRETED function "handle_win_input()" will be called. "skip_when_busy" determines whether handle_win_input can be called again when it has not finished the previous event yet.

***NOTE***

A new and better interface for application programs to show their interest in events has become available by means of the functions im_exposure_func() and im_input_func(). The support of the functions applic_exposure(), add_applic_exposure_func(), applic_win_input() and add_applic_win_input_func() is no longer guaranteed in future versions of SCIL_Image.

***RETURN VALUES***

None

***SEE ALSO***

add_applic_win_input_func add_applic_exposure_func applic_exposure

## *apply_spatial_bank*

## *apply_frequency_bank*

## *bank_frequency_response*

***NAME***

apply_spatial_bank, apply_frequency_bank, bank_frequency_response - perform filter banks

***SYNOPSIS***

```
#include "im_proto.h"

int apply_spatial_bank(IMAGE *in, IMAGE *bank, IMAGE *out, int begin,
int end)

int apply_frequency_bank(IMAGE *in, IMAGE *bank, IMAGE *out, int
begin, int end)

int bank_frequency_response(IMAGE *bank, IMAGE *out, int begin, int
end)
```

***DESCRIPTION***

These functions apply a number of filters stored in the slices of the 3D "bank" image to the 2D input image "in". The result of each filter is put into the corresponding slice of the 3D image "out".

apply_spatial_bank() performs the filtering by convolution().

apply_frequency_bank() performs the filtering by fast_hartley() or fast_fourier(), dependent on the bank image type (FLOAT_3D/ COMPLEX_3D).

bank_frequency_response() converts each spatial filter of the input bank to the frequency domain.

"begin" and "end" specify the filters (slices) that are to be applied. "end" is -1 specifies the filter at the highest Z-position of "bank".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

convolution  fast_hartley  fast_fourier

## *arbit_dilation*

***NAME***

arbit_dilation - dilation using an arbitrary shaped structuring element

***SYNOPSIS***

```
#include "im_proto.h"

int arbit_dilation(IMAGE *in, IMAGE *out, IMAGE *se, int bound)
```

***DESCRIPTION***

Performs a dilation on image "in" using structuring element "se" and stores the result in image "out". The structuring element "se" is a grey value image, with odd sizes in both directions, in which pixels encoded with a pixel value unequal to zero are part of the structuring element. The origin of the structuring element is the central pixel of the image. "bound" specifies if the pixels outside the image should be interpreted as foreground pixels ("bound" = 1) or as background pixels ("bound" = 0).

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

arbit_erosion  erosion3x3  dilation3x3

## *arbit_erosion*

***NAME***

arbit_erosion - erosion using an arbitrary shaped structuring element

***SYNOPSIS***

```
#include "im_proto.h"

int arbit_erosion(IMAGE *in, IMAGE *out, IMAGE *se, int bound)
```

***DESCRIPTION***

Performs an erosion on image "in" using structuring element "se" and stores the result in image "out". The structuring element "se" is a grey value image, with odd sizes in both directions, in which pixels encoded with a pixel value unequal to zero are part of the structuring element. The origin of the structuring element is the central pixel of the image. "bound" specifies if the pixels outside the image should be interpreted as foreground pixels ("bound" = 1) or as background pixels ("bound" = 0).

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

arbit_dilation  erosion3x3

---

## *area*

***NAME***

area - obtain area of object

***SYNOPSIS***

```
#include "im_aio.h"

long area(LIST *link)
```

***DESCRIPTION***

AIO primitive to obtain value of an object feature

area() returns the area of the object pointed to by "link". The area need not previously be specified with the measuring routine since it is always available after labeling with aio_label().

***RETURN VALUES***

area of the object in pixels on success
0 if link is not an object

***SEE ALSO***

measure  object_shape_meas  object_dens_meas

---

## *asctime*

## *clock*

## *ctime*

## *difftime*

## *gmtime*

## *localtime*

## *mktime*

## *strftime*

## *time*

***NAME***

asctime, clock, ctime, difftime, gmtime, localtime, mktime, strftime, time - time retrieval and conversion functions

***SYNOPSIS***

```
#include "time.h"

char *asctime(struct tm *tp)

long clock(void)

char *ctime(time_t *tp)

double difftime(time_t time2, time_t time1)

struct tm *gmtime(time_t *tp)

struct tm *localtime(time_t *tp)

time_t mktime(struct tm *tp)

size_t strftime(char *s, size_t smax, char *fmt, struct tm *tp)

time_t time(time_t *tp)
```

***DESCRIPTION***

These functions are interfaces to the standard C functions as implemented on the current system. The functionality of these function are:

asctime() converts the time in the structure "tp" into a string of the form:

```
      Thu Jul 24 12:08:09 1997
```

clock() returns the processor time used by the program since the beginning of execution, or -1 if unavailable. clock()/CLOCK_PER_SEC is a time in seconds.

ctime() converts the calendar time "tp" to local time; it is equivalent to:

```
      asctime(localtime(tp));
```

difftime() returns "time2" - "time1" expressed in seconds

gmtime() converts the calendar time "tp" into Coordinated Universal Time (UTC). It returns NULL if UTC is not available.

localtime() convert the calendar time "tp" into local time.

mktime() converts the local time in the struct "tp" into calendar time in the same representation used by time. The components will have values in the ranges shown. mktime() returns the calendar time of -1 if it cannot be represented.

time() returns the current calendar time or -1 if the time is not available.

strftime() formats the date and time information from "tp" into the string "s" according to the format string "fmt". The format string is analogous the printf() function. Ordinary characters (including the ’\0’) are copied into "s". Each %<character> is replaced as described below, using values appropriate for the local environment. No more than "smax" characters are placed into "s". strftime() returns the number of characters, excluding the ’\0’, or zero if more than "smax" characters were produced.

| | |
|---|---|
| %a | abbreviated weekday name |
| %A | full weekday name |
| %b | abbreviated month name |
| %B | full month name |
| %c | local date and time representation. |
| %d | day of the month (01-31) |
| %H | hour (24-hour clock) (00-23) |
| %I | hour (12-hour clock) (01-12) |
| %j | day of the year (001-366) |
| %m | month (01-12) |
| %M | minute (00-59) |
| %p | local equivalent of AM or PM |
| %S | second (00-61) |
| %U | week number of the year (Sunday as 1st day of week) (00-53) |
| %w | weekday (0-6, Sunday is 0) |
| %W | week number of the year (Monday as 1st day of week) (00-53) |
| %x | local date representation |
| %X | local time representation |
| %y | year without century |
| %Y | year with century |
| %Z | time zone name, if any |
| %% | % |

***STRUCTURES***

```
struct  tm {
    int tm_sec;         seconds after the minute (0,61)
    int tm_min;         minutes after the hour (0,59)
    int tm_hour;        hours since midnight (0,23)
    int tm_mday;        day of the month (1,31)
    int tm_mon;         months since January (0,11)
    int tm_year;        years since 1900
    int tm_wday;        days since Sunday (0,6)
    int tm_yday;        days since January 1 (0,365)
    int tm_isdst;       Daylight Saving Time flag
};
```

tm_isdst is positive if Daylight Saving Time is in effect, zero if not, and negative if the information is not available.

***RETURN VALUES***

See the description of each of the functions above.

---

## *atexit*

***NAME***

atexit - add program termination function

***SYNOPSIS***

```
#include <stdlib.h>

int atexit(void (*func)(void))
```

***DESCRIPTION***

This function is an interface to the standard C function as implemented on the current system. The functionality of this function is:

atexit() registers the function "func" to be called when the program terminates normally; it returns non-zero if the registration cannot be made.

***RETURN VALUES***

0 if successful
non-zero if the registration cannot be made.

***SEE ALSO***

exit _exit

---

## *atof*

## *atoi*

## *atol*

## *strtod*

## *strtol*

## *strtoul*

***NAME***

atof, atoi, atol, strtod, strtol, strtoul - convert ASCII to numbers

***SYNOPSIS***

```
double atof(char *nptr)

int atoi(char *nptr)

long atol(char *nptr)

double strtod(char *s, char **endp)

long strtol(char *s, char **endp, int base)

unsigned long strtoul(char *s, char **endp, int base)
```

***DESCRIPTION***

These functions are interface functions to the standard C functions as implemented on the current operating system. The functionality of these functions is:

These functions convert a string pointed to by "nptr" to floating point, integer, and long integer representation respectively. The first unrecognized character ends the string.

atof() recognizes an optional string of tabs and spaces, then an optional sign, then a string of digits optionally containing a decimal point, then an optional "e" or "E" followed by an optionally signed integer.

atoi() and atol() recognize an optional string of tabs and spaces, then an optional sign, then a string of digits.

strtod() converts the prefix of "s" to a double, ignoring leading white spaces. It stores a pointer to any  unconverted suffix in "*endp" unless "endp" is NULL. If the answer would overflow, HUGE_VAL is returned with the proper sign. If the answer would underflow, zero is returned. In either case "errno" is set to ERANGE.

strtol() converts the prefix of "s" to long, ignoring any leading white spaces. It stores a pointer to any unconverted suffix in "*endp" unless "endp" is NULL. If "base" is between 2 and 36, conversion is done assuming that the input is written in that base. If base is zero, the base is 8, 10, or 16; leading 0 implies octal and leading 0x or 0X hexadecimal. Letters in either case represent digits from 10 to base-1; a leading 0x or 0X is permitted in base 16. If the answer would overflow, LONG_MAX or LONG_MIN is returned, depending on the sign of the result, and errno is set to ERANGE.

strtoul() behaves like strtol() except that the result is unsigned long and the return value in case of overflow is ULONG_MAX.

***SEE ALSO***

scanf

## *auto_display*

## *don*

## *doff*

## *get_disp_mode*

***NAME***

auto_display, don, doff, get_disp_mode - enable/disable automatic display of images

***SYNOPSIS***

```
#include "im_infra.h"

int auto_display(int mode)

int don(void)

int doff(void)

int get_disp_mode(void)
```

***DESCRIPTION***

auto_display() turns on/off the automatic display of an image after each operation. If "mode" is 1 the images are displayed immediately after an operation is performed. "mode" is 0 disables displaying of images, even display_image() will not show the image. Disabling the automatic display of images is most often used to hide intermediate result of operations from the user's view.

don() is equivalent to auto_display(1), doff() is equivalent to auto_display(0).

get_disp_mode() can be used to retrieve the current display mode.

***NOTE***

The display mode set by these functions is a directive. It is up to the user-interface to decide if the mode is honored.

***RETURN VALUES***

auto_display(), don() and doff() return the previous display mode. So if automatic display was on, auto_display(0) would return 1.

get_disp_mode returns the current display mode, either 0 or 1.

## *auto_plane*

***NAME***

auto_plane - enable/disable automatic display of next plane

***PLATFORM***

UNIX, Macintosh.

***SYNOPSIS***

```
#include "disp_p.h"

int auto_plane(int flag)
```

***DESCRIPTION***

auto_plane() enables/disables the default behavior upon pressing the right mouse button inside an image window

When enabled, clicking the right mouse button inside an image that supports the next_plane() function results in a call to that function. If the pointer is pressed in the top of the window, the variable "num" (see next_plane()) gets a value of "1", in the middle a value of "0", and in the bottom "-1".

***RETURN VALUES***

None

***SEE ALSO***

auto_point  next_plane

## *auto_point*

***NAME***

auto_point - enable/disable automatic information on image pointing

***PLATFORM***

UNIX, Macintosh.

***SYNOPSIS***

```
#include "disp_p.h"

int auto_point(int flag)
```

***DESCRIPTION***

auto_point() enables/disables the default behavior upon pressing the left mouse button inside an in image window.

When auto_point is enabled, pressing the left mouse button inside an image-window results in showing the X and Y position and the value of the pixel at that position. The information is either displayed in a small floating window just below the mouse cursor, or in a small window just above the image.

The function call "point_im_display_buf("", 1)" can be used to let the information be printed in a window just below the cursor. (Only practical with fast display systems).

The function call "point_im_display_buf("", 0)" can be used to let the information be printed in a window just above the image (probably within the image title). (Practical on slower display systems).

It is recommended to disable the default behavior if AIO interactive object pointing is used. See point_im() and point_object().

***RETURN VALUES***

None

***SEE ALSO***

auto_plane  point_im_display_buf  handle_pim  point_im  point_object

## *average*

***NAME***

average

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See pix_average_val

---

## *b_to_comp*

***NAME***

b_to_comp - add buffered image to composite photo

***SYNOPSIS***

```
#include "image.h"
#include "silo.h"

int b_to_comp(COMPTR comptr, int sizex, int sizey, PIXEL *buf)
```

***DESCRIPTION***

| | | |
|---|---|---|
| comptr | - | Pointer to composite photo. |
| sizex | - | Part-image width. |
| sizey | - | Part-image height. |
| buf | - | Buffer containing the part-image. |

Function to append a part-image from a buffer to the composite photo.

***RETURN VALUES***

The position where the part-image went to:

| | | |
|---|---|---|
| x-start-coordinate | - | function value modulo 2048. |
| y-start-coordinate | - | function value div 2048. |

---

## *back_project*

***NAME***

back_project - Convert positions to original coordinates

***SYNOPSIS***

```
#include "grey_2dp.h"

int back_project(VAR_OBJECT *input, VAR_OBJECT *data, VAR_OBJECT
*output, int width)
```

***DESCRIPTION***

back_project() converts coordinates in the VAR_OBJECT "input" (1-dimensional), which are results of maximum_cost_path(), into coordinates in the original image (the image that was input to command resample_perp()). The VAR_OBJECT "data", produced by resample_perp(), is used for the backprojection. This var_object contains information about the way resample_perp() resampled the original image. The resampled image is the input to maximum_cost_path(). Its size in the first dimension is specified in "width". It is needed for the conversion, because the positions in var_object "input" are relative to the left hand side of the resampled image.

The obtained original coordinates are stored in the 2-dimensional var_object "output", in the row corresponding to the element of "input".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

maximum_cost_path maximum_trace resample_perp drawcurve

---

## *bangle*

***NAME***

bangle

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See angle_detection

---

## *base_name*

## *dir_name*

## *abs_pathname*

***NAME***

base_name, dir_name, abs_pathname - manipulate path-names

***SYNOPSIS***

```
#include "support.h"

void base_name(char *bname, const char *path)

void dir_name(char *dname, const char *path)

void abs_pathname(char *path)
```

***DESCRIPTION***

base_name() and dir_name() are used to isolate the file and directory part from a path-name. base_name() searches the given path-name "path" from the end for any directory or drive separator. When it finds one, all text following that separator is assumed to be a filename and is copied to the supplied buffer "bname". dir_name() also searches from the end of "path" for a directory or drive separator but on finding it, copies the part before the separator to "dname". Both buffers "bname" and "dname" are assumed to be of sufficient length to hold the returned name.

abs_pathname() takes a (possibly relative) path-name "path" of a file and translates that name (in place) into a full path-name, starting from the root of the filing-system. The length of the path-name (without the filename) should not exceed 256 bytes.

***RETURN VALUES***

None

---

## *baskel*

***NAME***

baskel

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See anchor_skelet

---

## *bcdist*

***NAME***

bcdist

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See constr_distance

---

## *bclose*

***NAME***

bclose

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See closing3x3

---

## *bcont*

***NAME***

bcont

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See contour

---

## *bcount*

***NAME***

bcount

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See pix_count

---

## *bdila*

***NAME***

bdila

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See dilation3x3

---

## *bdist*

***NAME***

bdist

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See distance

---

## *bdskel*

***NAME***

bdskel

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See dist_skelet

---

## *bedge*

***NAME***

bedge

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See set_border

---

## *bend*

***NAME***

bend - obtain bending energy of object

***SYNOPSIS***

```
#include "im_aio.h"

double bend(LIST *link)
```

***DESCRIPTION***

| | | |
|---|---|---|
| link | - | Link pointing to object |

AIO primitive to obtain value of an object feature

bend() returns the bending energy of the object pointed to by "link" if this has previously been measured.

***RETURN VALUES***

bending energy of object on success
0.0 if link is not an object or if bending energy has not been measured

***SEE ALSO***

measure  object_freeman_meas  object_shape_meas  object_dens_meas

## ***benke***

***NAME***

benke - texture segmentation filter search algorithm

***SYNOPSIS***

```
#include "im_proto.h"

int benke(IMAGE *pat1, IMAGE *pat2, IMAGE *out, int maxiter, double
gain, double convergence, int width, int height, int depth)
```

***DESCRIPTION***

Searches for a separation filter between the patterns "pat1" and "pat2". The search is finished when the energy ratio between "pat1" and "pat2" for an iteration differs no more than "convergence", or when "maxiter" iterations are performed. The convergence speed can be optimized with the "gain" value. The dimensions of the filter are given by "width", "height" and "depth". If one has trained the filter on "pat1" and "pat2", segmentation on images consisting of background texture "pat1" and object texture "pat2" can be obtained by applying convolution(), squaring the image and applying a smoothing (e.g. gauss) to measure local energy.

***LITERATURE***

K.K. Benke and D.R. Skinner, A direct search algorithm for global optimisation of multivariate functions, The Australian Computer Journal, vol. 23, no. 2, 1991, 73-85.

***EXAMPLE***

```
assumes two different texture in image A and B, and a composite image
in C:

int   s;
benke A B D 25 1.6 0.0005 5 5 1
s = (int)pix_abs_sum D
convolution C D A 0 s
mul_im A A B
gauss B B 5.0 5.0
```

***RETURN VALUES***

IE_OK (1) on succes

Negative error status on failure (see im_error.h)

***SEE ALSO***

convolution  pix_abs_sum  filter_energy_ratio

## *beros*

***NAME***

beros

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See erosion3x3

---

## *bin_disp_colors*

***NAME***

bin_disp_colors - choose colors for binary images display

***SYNOPSIS***

```
#include "disp_p.h"

int bin_disp_colors(int fg, int bg)
```

***DESCRIPTION***

bin_disp_colors() sets the fore- and background color used for displaying all binary images. By default the binary images are displayed using BLACK as the background and RED as the foreground color.

Only the eight primary colors are allowed, they are:

| | |
|---|---|
| BLACK | (0) |
| RED | (1) |
| GREEN | (2) |
| YELLOW | (3) |
| BLUE | (4) |
| MAGENTA | (5) |
| CYAN | (6) |
| WHITE | (7) |

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

---

## *binary_to_grey*

***NAME***

binary_to_grey - convert a binary image to a grey value image

***SYNOPSIS***

```
#include "bin_2dp.h"

int binary_to_grey(IMAGE *in, IMAGE *out, int val)
```

***DESCRIPTION***

The "in" image (a binary_2d image) is converted to the (grey_2d) "out" image. This means that for each "1" pixel in the binary image the corresponding pixel in the grey image is set to "val".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

binary_to_plane  plane_to_binary  set_im_type

---

## *binary_to_plane*

***NAME***

binary_to_plane - put a binary image in a bitplane of a grey image

***SYNOPSIS***

```
#include "bin_ 2dp.h"

int binary_to_plane(IMAGE *in, IMAGE *out, int plane)
```

***DESCRIPTION***

The "in" image (a binary_2d image) is put in the specified bitplane of the (grey_2d) "out" image.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

binary_to_grey  plane_to_binary  set_im_type

---

## *bit_ok*

***NAME***

bit_ok - check if a value is a binary value

***SYNOPSIS***

```
#include "im_infra.h"

int bit_ok(int value)
```

***DESCRIPTION***

"Value" must be either a "0" or a "1", if this is not the case, an error is generated and the following message is added to the error-stack:

Bit value [<value>] can only be (0 or 1)

***RETURN VALUES***

IE_OK (1) if the value is 0 or 1,
IE_NOT_OK (0) otherwise

***SEE ALSO***

range_ok  edge_ok

---

## *blabel*

***NAME***

blabel

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See label

---

## *blife*

***NAME***

blife

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See life

---

## *bline*

***NAME***

bline

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL-image routines.

See draw_line

---

## *blow*

***NAME***

blow - image blow-up

***SYNOPSIS***

```
#include "im_proto.h"

int blow(IMAGE *in, IMAGE *out, int hfact, int vfact, int dfact, int
adjust)
```

***DESCRIPTION***

Blow image "in" with a horizontal factor "hfact" a vertical factor "vfact" and a depth factor "dfact" (3d only), by repeating pixels and store the result in "out". The sizes of the output image "out" will be adjusted to fit the result only when the parameter "adjust" is true (non-zero).

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

fblow  reduce

---

## *bmaj*

***NAME***

bmaj

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See majority

---

## ***bopen***

***NAME***

bopen

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See opening3x3

---

## ***box_dimension***

***NAME***

box_dimension - texture measure, "fractal" dimension

***SYNOPSIS***

```
#include "image.h"

double box_dimension(IMAGE *input, IMAGE *mask, int fsizemin, int
fsizestep, int fsizemax)
```

***DESCRIPTION***

Reduce the image "input" to smaller resolutions, the reduction factors "filtersize" ranges from "fsizemin" to "fsizemax" in steps of "fsizestep". In each reduction step, a block of "filtersize"*"filtersize" is reduced to one pixel in an intermediate image, this pixel is given a value equal to the difference between the minimum and maximum in the original block, divided by "filtersize".

For each resolution (step) the average of the intermediate image is calculated.

Finally, a Least Square Fit is done through the log(filtersize) log(average) plot. The angle of the plot is the texture measure.

The calculation of the texture is only done in the areas where the bit-image "mask" has value 1.

***RETURN VALUES***

The texture value is returned. In case of error, this is 0.

***SEE ALSO***

gld_mean  gld_entropy  gld_contrast  gld_asymmetry
glc_entropy  glc_contrast  glc_asymmetry  glr_nonuniformity
glr_shortrunemphasis  glr_longrunemphasis  glr_greynonuniformity
glr_percentage  edge_average  dist_average

---

## *bperc*

***NAME***

bperc

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See percentile

---

## *bprop*

***NAME***

bprop

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See propagation

---

## *bpsr*

***NAME***

bpsr

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See psremoval

---

## *bremh*

***NAME***

bremh

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See remove_holes

---

## ***bril***

## ***stop_bril***

***NAME***

bril, stop_bril - interactive part image processing

***DESCRIPTION***

This is an old function name, please use the function lens() and stop_lens()

See lens

---

## ***bsearch***

## ***qsort***

***NAME***

bsearch - binary search of a sorted table

qsort - quick sort

***SYNOPSIS***

```
void *bsearch(void *key, void *base, size_t n, size_t size, int
(*cmp)(void *key, void *datum))

void qsort(void *base, size_t n, size_t size, int (*cmp)(void *, void
*))
```

***DESCRIPTION***

This function is an interface to the standard C function as implemented on the current system. The functionality of this function is:

bsearch() searches "base[0]" … "base[n]" for an item that matches "*key". The function "cmp" must return negative if  its first argument ( the search key) is less than its second (a table entry), zero if equal, and positive if greater. Items in the array "base" must be in ascending order. bsearch() returns a pointer to a matching item, or NULL if none exists.

qsort() sorts in ascending order "n" elements of an array "base[0]" … "base[n]" of objects of size "size". The comparison function cmp is as in bsearch().

***RETURN VALUES***

bsearch returns a pointer to a matching item, or NULL if none exists
qsort returns nothing

---

## *bskbp*

***NAME***

bskbp

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See skelpoints

---

## *bskel*

***NAME***

bskel

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See hild_skelet

---

## *bskep*

***NAME***

bskep

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See skelpoints

---

## *bsklp*

***NAME***

bsklp

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See skelpoints

---

## *bsngl*

***NAME***

bsngl

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See single_pixels

---

## *buf_from_silo*

***NAME***

buf_from_silo - retrieve silo image into buffer

***SYNOPSIS***

```
#include "silo.h"

int buf_from_silo(SILOPTR siloptr, int silo_key, PIXEL *buf)
```

***DESCRIPTION***

| | | |
|---|---|---|
| siloptr | - | Pointer to the image-silo. |
| silo_key | - | Numerical entry in the silo. |
| buf | - | Buffer for the data. |

Copies the silo_key image from the image-silo to a buffer.

***NOTE***

This is the general silo output function. Other routines like part_from_silo and im_from_silo are using this routine to access the silo.

***RETURN VALUES***

IE_OK (1) on succes
Negative error status on failure (see im_error.h)

***SEE ALSO***

part_from_silo  im_from_silo

---

## *buf_to_silo*

### *NAME*

buf_to_silo - store buffered image in image-silo

### *SYNOPSIS*

```
#include "image.h"
#include "silo.h"

int buf_to_silo(SILOPTR siloptr, int silo_key, PIXEL *buf, int sizex,
int sizey)
```

### *DESCRIPTION*

| | | |
|---|---|---|
| siloptr | - | Pointer to the image-silo. |
| silo_key | - | Numerical entry in the silo. |
| buf | - | Buffer from which the data is taken. |
| sizex | - | Virtual width of the image data. |
| sizey | - | Virtual height of the image data. |

Transfers the image data from a buffer to an image-silo.

### *NOTE*

This is the general silo input function. Other routines like part_to_silo and im_to_silo are using this routine to access the silo.

### *RETURN VALUES*

IE_OK (1) on succes
Negative error status on failure (see im_error.h)

### *SEE ALSO*

part_to_silo  im_to_silo

## *calc_greyvalue*

## *RGB_clear_extra*

***NAME***

calc_greyvalue - calculate the grey value in a RGB color image

RGB_clear_extra - clear the extra field in a RGB color image

***SYNOPSIS***

```
#include "color_2dp.h"

int calc_greyvalue(IMAGE *image)

int RGB_clear_extra(IMAGE *image)
```

***DESCRIPTION***

calc_greyvalue() takes the image "image" and when it is an RGB image the grey-value of each pixel is stored in the 4th byte of the RBG structure (the "extra" field). The RGB structure is defined in "image.h ".

The grey-value is calculated according to:

```
      greyval = 0.587 * green + 0.114 * blue + 0.299 * red
```

RGB_clear_extra() clears the fourth channel in a RGB color image "image", the "extra" field of
all pixels is set to 0.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

convert

## *calibrated_density*

***NAME***

calibrated_density - object size and calibrated density measurement

***SYNOPSIS***

```
#include "im_proto.h"

int calibrated_density(IMAGE *label_im, IMAGE *grey_im, char *fname,
int append, VAR_OBJECT *table)
```

***DESCRIPTION***

Measure objects in the image "grey_im" using the labeled image "label_im" as an object indicator and write these parameters in a table on the console or in a text file.

For each object present int the image "label_im", the corresponding object pixels in the image "grey_im" are used for the measurement. The measured parameters are:

* the coordinates of the object’s center of gravity
* the total calibrated object density. This density measurement can be influenced by the parameter "table":
  - if "table" is specified, the original pixel values of "grey_im" are used as an index in the FLOAT table "table". The value which is found at the corresponding position is added to the total density for the corresponding object.
  - if "table" is not specified (NO_TABLE) the original pixel value is added to the total density for the corresponding object.
* the object size in number of pixels
* the ratio of the object density and the object size (average pixel value within the object).

The measured parameters are written to the text file "fname" if specified, or printed on the controlling terminal ("fname" = NULL). If "append" is set (=1) then the generated table will be appended to the text file "fname" (if specified and existing). If "fname" already exists and "append" is not set (=0) then the file "fname" will be overwritten.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

density  label  shape

## *canny*

***NAME***

canny - Canny edge detector

***SYNOPSIS***

```
#include "im_proto.h"

int canny(IMAGE *in, IMAGE *out, IMAGE *Lx, IMAGE *Ly, double sigma,
double acc, int fwidth, int nonmax)
```

***DESCRIPTION***

Performs edge detection based on the Canny algorithm.

The output image contains the magnitude of the gradient vector. In case "Lx" and "Ly" are valid images, the magnitude of the gradient in the X- and Y-direction are stored there. "Lx" is the gradient in the X-direction, obtained by calling:

```
fuzzy_derivative(in, Lx, sigma, sigma, 1, 0, acc, acc, fwidth,
fwidth)
```

"Ly" is the gradient in the Y-direction, obtained by calling:

```
fuzzy_derivative(in, Ly, sigma, sigma, 0, 1, acc, acc, fwidth,
fwidth)
```

Optionally the non-maximal values in the direction of the gradient in a 3x3 neighborhood are suppressed ("nonmax" is 1). When the suppression is on, the corresponding pixels in the "Lx" and "Ly" image are cleared.

Since the magnitude of the gradient vector in the output image "out" is based on "Lx" and "Ly" the true sigma of the Gaussian applied to the input image is equal to sqrt(2) times the given "sigma".

***LITERATURE***

"A Computational Approach to Edge Detection", Canny, J., IEEE PAMI, vol.8, no.6, pages 679-698, November 1986

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

gauss  fuzzy_derivative  prewitt_diff  roberts_diff  sobel_diff

## *cdens*

***NAME***

cdens

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See calibrated_density

---

## *chain*

***NAME***

chain - load a C source code file and execute

***SYNOPSIS***

```
chain <filename>
```

***DESCRIPTION***

The file "filename" is loaded in the interpreter of SCIL and the execution is started at the main() function. If no main() function is found, an error message will be generated.

***EXAMPLE***

```
[C1] chain demo.c
```

***SEE ALSO***

load  run

---

## *chaincode_to_image*

***NAME***

chaincode_to_image - convert chain-code list into labeled image

***SYNOPSIS***

```
#include "grey_2dp.h"

int chaincode_to_image(VAR_OBJECT *input, IMAGE *image)
```

***DESCRIPTION***

The list of chain-code object representations given in the VAR_OBJECT "input" is converted into objects in a labeled image, "image". The list is assumed to be structured like the output of  image_to_chaincode(). As the original grey-values of the objects are  not in the list, the objects are given a grey-value equal to their sequence number in the list, starting with 1.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

image_to_chaincode  chaincode_to_xy  put_xy_into_image

## *chaincode_to_xy*

***NAME***

chaincode_to_xy - convert chain-codes into coordinates

***SYNOPSIS***

```
#include "grey_2dp.h"

int chaincode_to_xy(VAR_OBJECT *input, VAR_OBJECT *output, int
offset)
```

***DESCRIPTION***

The chain-code string in "input" is converted into a VAR_OBJECT "output" with (x,y)-coordinate pairs. A chain-code string is a one dimensional VAR_OBJECT, containing:

> the x-coordinate of the first pixel, the y-coordinate of the first pixel, the number of chain-codes used for description of the curve, the chain-codes (see image_to_chaincode())

This is the representation of one curve as generated by image_to_chaincode(). The list generated by image_to_chaincode() in its output variable "output" describes a set of objects, each constituted of one or more curves. To convert a curve which is part of an object, "offset" must be point to that curve. "offset" is a number-offset, NOT a object-offset, NOR a curve-offset.

The offset to the first curve in "input" is 2:

> the first number (offset 0) is the object count in "input" and the second number (offset 1) is the curve count of the first object, so "offset = 2" points to the start of the representation of the first curve of the first object.

The offset for other objects and curves depends upon the length of the preceding curves (number of freeman codes), the number of curves in preceding objects and the number of preceding objects.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

image_to_chaincode  chaincode_to_image  put_xy_into_image

## *change_image_size*

***NAME***

change_image_size - change the dimensions of an image

***SYNOPSIS***

```
#include "im_proto.h"

int change_image_size(IMAGE *im, int width, int height, int depth)
```

***DESCRIPTION***

change_image_size() changes the dimensions of the image"im". The old image content is lost by this operation except when the image already is of the given sizes. The new image has dimensions "width"*"height"*"depth".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

set_im_type

## *channel_bi_threshold*

***NAME***

channel_bi_threshold - multi-channel bi-level threshold operation

***SYNOPSIS***

```
#include "im_proto.h"

int channel_bi_threshold(IMAGE *in, IMAGE *out, double min1, double
max1, double min2, double max2, double min3, double max3, double
min4, double max4)
```

***DESCRIPTION***

channel_bi_threshold() performs a bi-level threshold operation on all channels of a multi-channel image "in" and stores the result in the binary image "out". For each of the channels (maximum of  4 channels), the pixels are compared with the range for that channel ("min1" .. "max1" for the first channel etc.). Only if the values for each of the channels of a pixel are inside the range (threshold values included) for that channel, the corresponding pixel in the output image is set to "1" (foreground pixel). Otherwise the pixel in the output image is set to "0" (background pixel).

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

bi_threshold

## *chdir*

***NAME***

chdir - change current working directory

***PLATFORM***

UNIX, MS-Windows.

***SYNOPSIS***

```
int chdir(char *dirname)
```

***DESCRIPTION***

This function is an interface to the standard C function as implemented on the current system. The functionality of this function is:

dirname is the address of the pathname of a directory, terminated by a null byte. chdir causes this directory to become the current working directory.

***RETURN VALUES***

Zero is returned if the directory is changed; -1 is returned if the given name is not that of a directory or is not readable by the user.

***SEE ALSO***

cd

## *check_image_integrity*

***NAME***

check_image_integrity - check images on their integrity

***SYNOPSIS***

```
#include "im_infra.h"

int check_image_integrity(int print)
```

***DESCRIPTION***

check_image_integrity() checks all images on the integrity and repairs them if necessary. When "print" is set (=1), the function prints what it is doing and the irregularitis it finds. If "print" is not set (=0), the function keeps silent abouts its actions.

The function checks if the input and output field of the IMAGE structure point to the same image type descriptor. If these are different, it will remove the output type descriptor and throw away the image that was linked to the output. Next the output is connected to the same descriptor as the input.

This is meant for situations where an operation is aborted after a call to pre_op(). In such a situation the newly created output image became invalid because the operation was aborted. The image is restored to the state it was in before the operation began.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

pre_op

## *check_status*

***NAME***

check_status - produce an alert box with a warning

***SYNOPSIS***

```
#include "im2scil.h"

void check_status(int status, char *str)
```

***DESCRIPTION***

If "status" is unequal to zero, this routine produces an alert box in the middle of the screen containing information about what error occurred. "Status" is an error code as they are defined in the include file "im_error.h". For each of the defined values a default message will be printed. Also the global string variable "ErName"will be printed. "ErName" contains the name of the function, check_status() was called from or is closely related to. If the string "str" is not empty it will also be printed in the alert box.

This alert box can only be removed by pressing either of the two buttons in the bottom of the box. The buttons have only a different result when an interpreted program or a macro file is being executed.

| | |
|---|---|
| [Continue] | allows the interpreter to carry on with the remainder of the interpeted program or macro file. |
| [Stop] | tells the interpreter to stop executing the program or macro file. |

After one of the two buttons has been pressed, a call to check_image_integrity() is automatically done.

***NOTE***

This function is only present for compatibility with older version of SCIL_Image. As of SCIL_Image 1.4 a more flexible error-handling mechanism has been implemented. Please refer to the "User’s Manual" for more information. Further usage of check_status() is strongly discouraged.

***RETURN VALUES***

None

***SEE ALSO***

check_image_integrity

## *clear_im*

## *clear_part_image*

***NAME***

clear_im, clear_part_image - clear image

***SYNOPSIS***

```
#include "im_proto.h"

int clear_im(IMAGE *out)

int clear_part_image(IMAGE *im, int sx, int sy, int sz, int width,
int height, int depth)
```

***DESCRIPTION***

clear_im() clears the image "out" (set all pixels to 0).

clear_part_image clears the part of the image "im" with dimensions "width" * "height" * "depth" starting at the position ("sx", "sy", "sz").

***NOTE***

For more powerful image arithmetic expressions (scaling, adding offsets, etc.), use the function eval().

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

set_int

## *clear_var_object*

***NAME***

clear_var_object - clear the contents of a var_object

***SYNOPSIS***

```
#include "objectsp.h"

int clear_var_object(VAR_OBJECT *object)
```

***DESCRIPTION***

Clear the contents of a var_object "object" by setting its entire piece of memory to 0, thus effectively setting all elements of the var_object to 0. The type and sizes of the object are not altered.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

var_object

## *clearerr*

## *feof*

## *ferror*

***NAME***

clearerr, feof, ferror - error functions

***SYNOPSIS***

```
void clearerr(FILE *stream)

int feof(FILE *stream)

int ferror(FILE *stream)
```

***DESCRIPTION***

This function is an interface to the standard C function as implemented on the current system. The functionality of this function is:

clearerr() clears the end-of-file and error indicators for "stream".

feof() returns non zero if the end-of-file indicator for "stream" is set.

ferror() returns non-zero if the error indicator for "stream" is set.

***RETURN VALUES***

See description of functions

## *clip*

***NAME***

clip - image clipping

***SYNOPSIS***

```
#include "im_proto.h"

int clip(IMAGE *in, IMAGE *out, int lowest, int highest)
```

***DESCRIPTION***

Clip the pixel values from image "in" between the values "lowest" and "highest", i.e. substitute pixel values less than "lowest" by value "lowest" and greater than "highest" by value "highest", and store the result in image "out". All other pixel-values are copied directly from "in" to "out"

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

threshold  contrast_stretch  equalize  tri_state_threshold  zlookup

---

## *close*

***NAME***

close - close a file

***SYNOPSIS***

```
int close(int fildes)
```

***DESCRIPTION***

This function is an interface to the standard C function as implemented on the current system. The functionality of this function is:

Given a file descriptor such as returned from an open() or creat() call, close() closes the associated file. A close() of all files is automatic on exit, but since there is a limit on the number of open files per process, close() is necessary for a program which deals with many files.

***RETURN VALUES***

| | |
|---|---|
| 0 | is returned if a file is closed; |
| -1 | is returned for an unknown file descriptor. |

***SEE ALSO***

creat  open

---

## close_comp

***NAME***

close_comp - close composite photo.

***SYNOPSIS***

```
#include "silo.h"

void close_comp(COMPTR comptr)
```

***DESCRIPTION***

| | | |
|---|---|---|
| comptr | - | Pointer to composite photo. |

Function to close a composite photo and returns allocated space to the system.

***RETURN VALUES***

None

## close_silo

***NAME***

close_silo - close a silo file

***SYNOPSIS***

```
#include "silo.h"

int close_silo(SILOPTR siloptr)
```

***DESCRIPTION***

| | | |
|---|---|---|
| siloptr | - | Pointer to an image-silo. |

Closes an image-silo and returns allocated space to the system.

***RETURN VALUES***

Always IE_OK (1)

## *closing3x3*

***NAME***

closing3x3 - close

***SYNOPSIS***

```
#include "im_proto.h"

int closing3x3(IMAGE *in, IMAGE *out, int iter, int con, int bound)
```

***DESCRIPTION***

Performs a closing from image "in" to image "out", which is performed by "iter" dilations from "in" to "out" followed by "iter" erosions from "out" to "out". The operation deletes holes and background parts having a width less than two times the specified number of iterations "iter".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

opening3x3  erosion3x3  dilation3x3

---

## *clut_by_name*

***NAME***

clut_by_name - get pointer of clut by its name

***SYNOPSIS***

```
#include "im_infra.h"

CLUT *clut_by_name(char *name, int case_sensitive)
```

***DESCRIPTION***

If the pointer to an clut is not at hand you obtain that pointer by use of this function. "name" is the name of the clut, "case_sensitive" specifies whether a distinction between lower case and upper case characters should to be made. If zero then no distinction is made.

***RETURN VALUES***

Pointer the clut on success or
NULL if clut "name" does not exist

***SEE ALSO***

create_clut  destroy_clut  set_clut

---

## *clut_ok*

## *is_clut*

***NAME***

clut_ok - check if the supplied pointer is a valid clut pointer

is_clut - tell if the supplied pointer is a clut (no warning)

***SYNOPSIS***

```
#include "im_infra.h"

int clut_ok(CLUT *clut)

int is_clut(CLUT *clut)
```

***DESCRIPTION***

The pointer "clut" is checked if it points to a valid clut. The linked list in which all the cluts are present is scanned for the occurrence of "clut". If no clut exist with this pointer, an error is generated and the following message is added to the error-stack:

Non existing clut pointer.

The function is_clut() performs the same check and has the same return values but does not generate an error. The function is meant for testing purposes.

***RETURN VALUES***

IE_OK (1) if the pointer is a valid clut.
IE_NOT_OK (0) if the pointer is not an clut.

***SEE ALSO***

create_clut  destroy_clut  set_clut

## *cmp_pixels*

***NAME***

cmp_pixels - image comparison

***SYNOPSIS***

```
#include "im_proto.h"

int cmp_pixels(IMAGE *in1, IMAGE *in2, VAR_OBJECT *first)
```

***DESCRIPTION***

Compare image "in1" and "in2" pixel by pixel and record the number of different pixels. The address of the first different pixel is stored in the object "first".

***RETURN VALUES***

the number of different pixels

---

## *cnvo*

***NAME***

cnvo

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See convolution

---

## *color_get_model_size*

## *color_set_color_model*

## *prefered_color_model*

## *set_color_model*

***NAME***

color_get_model_size - get the size of one pixel of  a color-model

color_set_color_model - set the color model of a color image

prefered_color_model - indicate the prefered color-model for an image

set_color_model - set the color model of a color image

***SYNOPSIS***

```
#include "color2dp.h"

int color_get_model_size(int model)

int color_set_color_model(IMAGE *im, int inout, int model)

int prefered_color_model(IMAGE *im, int inout, int model)

int set_color_model(IMAGE *im, int model)
```

***DESCRIPTION***

color_get_model_size() returns the size of one pixel of the color-model "model". The size is returned in the number of bytes for one pixel. For a list of the currently support color-models see convert_cmodel().

color_set_color_model() changes the color-model model of image "im" to "model", the "inout" parameter specifies whether to change the in-descriptor image or the out-descriptor image. The difference between the in and out-descriptor image is explained int the "User’s manual" in the chapter "Programming with Image". "inout" = 0 means the in-descriptor; "inout" = 1 means the out-descriptor.

prefered_color_model() is used to signal to the pre_op() function that the color-model for the output (color)-image "im" in the following pre_op() call should be (COMPARE mode) or should become (ADJUST mode) "model". The "inout" parameter is equal to the "inout" parameter of color_set_color_model(). At the moment however, changing the input-descriptor is not supported by pre_op(), so it should always be set to "1".

set_color_model() also changes the color-model of image "in" to "model", but it always changes the in-descriptor image. This function should therefore not be used on a color-image after a call to pre_op().

***NOTE***

Both set_color_model() and color_set_color_model() only change the color-model of the image, the data of the image is lost. Except when the image was already off the correct model.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

convert_cmodel  pre_op

---

## *com_dialog*

***NAME***

com_dialog - activate dialog box from a command string

***SYNOPSIS***

```
#include "md_gen.h"

void com_dialog(char *string)
```

***DESCRIPTION***

| | | |
|---|---|---|
| char *string | - | complete or partial command string |

com_dialog() can be used to activate a dialog box by giving a complete or partial command through the parameter string. The command must be present in the command description file of SCIL. It can be used to activate dialog boxes from within compiled or interpreted functions and allows dialog boxes with default values other than specified in the command description file.

***NOTE***

The dialog boxes of SCIL_Image are "modeless", this means that the com_dialog() function returns almost immediately and the program continues to run. The dialog box itself stays on screen until it is removed by means of user-interaction.

***EXAMPLE***

```
com_dialog("readf cermet");
```

***RETURN VALUES***

None

---

## *compact_silo*

***NAME***

compact_silo - removes gaps from an image-silo

***SYNOPSIS***

```
#include "silo.h"

void compact_silo(SILOPTR siloptr)
```

***DESCRIPTION***

| | |
|---|---|
| siloptr - | Pointer to an image-silo. |

Removes gaps from an image-silo. Gaps can occur when images are deleted from the silo. Images in the sile are shifted towards the beginning of the silo to close all existing gaps.

***RETURN VALUES***

None

---

## *complex_im*

## *make_complex_im*

***NAME***

complex_im, make_complex_im - convert two floating images into a complex image

***SYNOPSIS***

```
#include "im_proto.h"

int complex_im(IMAGE *in1, IMAGE *in2, IMAGE *out)

int make_complex_im(IMAGE *in1, IMAGE *in2, IMAGE *out)
```

***DESCRIPTION***

Store the elements of image "in1" in the real part of the complex image "out", and the elements of image "in2" in the imaginary parts of the image "out".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

real_im imaginary_im

---

## *complx*

***NAME***

complx

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See complex_im

---

## *compute_clut*

***NAME***

compute_clut - map the colors of a new lookup table to the system’s

***SYNOPSIS***

```
#include "disp_p.h"

int compute_clut(CLUT *clut)
```

***DESCRIPTION***

When rgb triplets are supplied in a clut by the user, this routine will calculate which entry in the system’s table is the best fit for each triplet. The entry will be stored in the array "table" of the CLUT structure "clut". These values then will be directly send to the display of the image when the clut is attached to a display window of an image.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

create_clut  set_clut

---

## *con_ok*

***NAME***

con_ok - check if the connectivity parameter is correct.

***SYNOPSIS***

```
#include "im_infra.h"

int con_ok(int con)
```

***DESCRIPTION***

Various operations have a connectivity parameter that must have either of the values 4 or 8. This functions checks if "con" is indeed one of the two. In case "con" is not the correct value an error is generated and the following message is added to the error-stack:

```
      Connectivity [<con>] can only take values 4 or 8
```

The special connectivity of first 8 connected and then 4 connected or vice versa, which is also used in image processing is checked with the function "con6_ok".

***RETURN VALUES***

IE_OK (1) if "con" is 4 or 8
IE_NOT_OK (0) if "con" has any other value

***SEE ALSO***

con6_ok

## *con6_ok*

***NAME***

con6_ok - check connectivity parameter on 4, 8, 48 or 84

***SYNOPSIS***

```
#include "im_infra.h"

int con6_ok(int con)
```

***DESCRIPTION***

Various operations have a connectivity parameter that must have one of the values 4, 8, 48 or 84. This function checks if "con" is indeed one of these. In case "con" is not one of the valid values, an error is generated and the following message is added to the error-stack:

```
      Connectivity [<con>] can only take values 4,8,48,84
```

The connectivity of just 4 or 8 is checked with the function "con_ok".

***RETURN VALUES***

IE_OK (1) if "con" is 4, 8, 48 or 84
IE_NOT_OK (0) if "con" is any other value

***SEE ALSO***

con_ok

---

## *conjugate_im*

***NAME***

conjugate_im - the conjugate value of each image element

***SYNOPSIS***

```
#include "im_proto.h"

int conjugate_im(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

For each complex element a+bi of "in" compute the conjugate a-bi and store the result in "out"

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

---

## *conjugate_mul_im*

***NAME***

conjugate_mul_im - complex conjugate multiplication

***SYNOPSIS***

```
#include "im_proto.h"

int conjugate_mul_im(IMAGE *in1, IMAGE *in2, IMAGE *out)
```

***DESCRIPTION***

For each complex element a+bi of "in1" and c+di of "in2", calculate the conjugate multiplication (a-bi) * (c+di) and store the result in "out".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

conjugate_im  mul_im

## *constr_distance*

***NAME***

constr_distance - constrained distance transformation

***SYNOPSIS***

```
#include "im_proto.h"

int constr_distance(IMAGE *in, IMAGE *constraint, IMAGE *out, int
hstep, int dstep int kstep, int b)
```

***DESCRIPTION***

Apply a constrained distance transformation to the binary image "in" and store the resulting distance values into the corresponding pixels of the grey valued image "out". The constrained distance transform is a special variant of distance transformation (see "distance"). To the transformation, conditions are added, as specified by means of the grey values image "constraint". Non-zero points within the image "constrained" are interpreted as preset values for the distance transform values of the corresponding pixels, regardless of the result of the distance transform itself.
Distances between two points are locally increased by the preset pixels, if the preset pixels are on the shortest path between the two points.
This is a way of introducing obstacles: an infinite preset value (in practice a large integer, say 32000) acts as an absolute obstacle, and the nearest shortest path will be searched.

The way the distances between the pixels are defined is specified by the parameters "hstep", "dstep" and "kstep".For the description of these parameters and the "b" parameter, see distance()

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

distance

## *contour*

***NAME***

contour - contour detection

***SYNOPSIS***

```
#include "im_proto.h"

int contour(IMAGE *in, IMAGE *out, int edge, int conn, int obj_bkg)
```

***DESCRIPTION***

Detects the object contours in image "in" and stores the result in image "out". The operation is executed by removing all the object pixels that do not belong to the outer contour of the objects, while preserving the connectivity "con" between all the resulting contour pixels of one object. "obj_bkg" specifies whether the contour of the object is determined on or the contour of the background. When "edge" is set (=1), the border around the images is set before the operation.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

---

## *contrast_stretch*

***NAME***

contrast_stretch - contrast stretch

***SYNOPSIS***

```
#include "im_proto.h"

int contrast_stretch(IMAGE *in, IMAGE *out, double lperc, double
hperc)
```

***DESCRIPTION***

Linear contrast stretching based upon the histogram of the pixel values in the image "in". The histogram of pixel values in image "in" is calculated. Then two pixel values (lowval and highval) are assigned to the cumulative histogram percentiles "lperc" and "hperc". Now the pixels in "in" are linearly rescaled is such a way that lowval corresponds with the smallest (0) and highval with the largest possible value (255). The result is stored in "out".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

clip  threshold  equalize  tri_state_threshold  lookup

---

## convert

### NAME

convert - convert an image into another image type

### SYNOPSIS

```
#include "im_infra.h"

int convert(IMAGE *in, IMAGE *out, int out_type)
```

### DESCRIPTION

Convert an image into another type. If "out_type" is not specified then the type of the image "out" is the type the data of image "in" is converted into. If "out_type" is specified, image "out" will become of type "out_type" and again the data is converted into that type. The available types (see image.h) are:

GREY_2D
BINARY_2D
FLOAT_2D
COMPLEX_2D
GREY_3D
BINARY_3D
FLOAT_3D
COMPLEX_3D
COLOR_2D
COLOR_3D
LABEL_2D
LABEL_3D

### RETURN VALUES

IE_OK (1) on success
Negative error status on failure (see im_error.h)

## *convert_cmodel*

***NAME***

convert_cmodel - convert a color-image into another color-model

***SYNOPSIS***

```
#include "color2dp.h"

int convert_cmodel(IMAGE *in, IMAGE *out, cmodel)
```

***DESCRIPTION***

convert_cmodel() converts the data of a color-image "im" into another color-model "cmodel" in image "out". The currently implemented color-models are (see also image.h):

| model | model name in Image | value |
|---|---|---|
| RGB | RGB_T | 1 |
| CIE XYZ | XYZ_T | 2 |
| CIE L*a*b* | CIELAB_T | 3 |
| HSI | HSI_T | 4 |
| CMYK | CMYK_T | 5 |

Specifying 0 for the color-model, means taking the color-model of the output image (if it is already a color-image otherwise RGB_T is taken).

Currently the conversion of the color-models are implemented as:

RGB to XYZ:
```
| X |   | m11 m12 m13 | | R |
| Y | = | m21 m22 m23 | | G |
| Z |   | m31 m32 m33 | | B |
```

for m11 .. m33 see set_matrix_type()

RGB to CMYK
```
C = 1.0 - R
M = 1.0 - G
Y = 1.0 - B
K = minimum( C, Y, K )
C = C - K
M = M - K
Y = Y - K
```

RGB to HSI:
```
I = (R + G + B )/3.0
S = 1.0 - 3.0 * minimum( R, G, B )/( R + G + B)
if  G > B:
  H = acos((0.5*(R-G+R-B))/sqrt((R-G)*(R-G)+(R-B)*(G-B)))
else
  H = 2*PI- acos((0.5*(R-G+R-B))/sqrt((R-G)*(R-G)+(R-B)*(G-B)))
```

XYZ to L*a*b*:
```
L* = cbrt(116 * (Y/Yn))-16      for  Y/Yn > 0.008856
L* = 903.3 * (Y/Yn)             for  Y/Yn <= 0.008856
```

```
a* = 500 * [F(X/Xn)-F(Y/Yn)]
b* = 200 * [F(Y/Yn)-F(Z/Zn)]

for any the ratios of X/Xn, Y/Yn, Z/Zn in a* and b* :
F(X/Xn) = cbrt(X/Xn)                for X/Xn > 0.008856
F(X/Xn) = 7.787 * (X/Xn) + 16/116   for X/Xn <= 0.008856
```

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

color_set_color_model  set_color_model

---

## *convolution*

***NAME***

convolution - convolution

***SYNOPSIS***

```
#include "im_proto.h"

int convolution(IMAGE *in, IMAGE *conv, IMAGE *out, int addval, int
divval)
```

***DESCRIPTION***

Apply a one- or two-dimensional convolution filtering to image "in" and store the result in image "out". The coefficients for the convolution should be specified by image "conv". The dimensions of the convolution mask are given by the dimensions of image "conv". This mask is not necessarily square. Optionally the result may be rescaled: the constant "addval" is added to the convolution result, after which it is divided by "divval".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

---

## *cooccur*

***NAME***

cooccur - calculate co-occurrence matrix

***SYNOPSIS***

```
#include "im_proto.h"

int cooccur(IMAGE *in, IMAGE *out, int xdist, int ydist)
```

***DESCRIPTION***

Calculates the co-occurrence matrix (second order statistics) of the input image on distance "xdist" and "ydist", and stores the result in "out". The input image greylevel range must be between 0 and 255.

***LITERATURE***

R.M. Haralick, K. Shanmugan and I. Dinstein, Textural features for image classification, IEEE trans. SMC, vol. 3, 1973, 610-621.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

---

## *copy_channel*

***NAME***

copy_channel - copy a channel from one image to another

***SYNOPSIS***

```
#include "im_infra.h"

int copy_channel(IMAGE *in, IMAGE *out, int inchan, int outchan)
```

***DESCRIPTION***

copy_channel() copies the image data from a channel from image "in" to a channel in image "out". The numbers of the channels are specified by "inchan" for the input image and "outchan" for the output image. Channels are numbered from 0 and all image types have at least one channel.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

split_channels  join_channels

---

## *copy_clut*

***NAME***

copy_clut - copy the contest of a CLUT to another CLUT

***SYNOPSIS***

```
#include "im_infra.h"

CLUT *copy_clut(CLUT *source, CLUT *dest, char *name)
```

***DESCRIPTION***

Copies the contents of the CLUT "source" to the CLUT "dest". If "dest" is a NULL pointer, a new CLUT is created with the name "name".

***RETURN VALUES***

Pointer to the destination CLUT or

| | |
|---|---|
| NULL if: | -"source" is not a CLUT |
| | -"dest" is not NULL and not a clut |
| | -No memory available to create a new clut or invalid name given. |

***SEE ALSO***

create_clut

---

## *copy_im*

***NAME***

copy_im - copy image

***SYNOPSIS***

```
#include "im_proto.h"

int copy_im(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

Copy image "in" to image "out".

***NOTE***

For more powerful image arithmetic expressions (scaling, adding offsets, etc.), use the function eval().

***RETURN VALUES***

IE_OK (1) on success

Negative error status on failure (see im_error.h)

***SEE ALSO***

copy_part_image

---

## *copy_object*

***NAME***

copy_object - copy object from source to destination image

***SYNOPSIS***

```
#include "im_aio.h"

int copy_object(IMAGE *src_im, IMAGE *dst_im, LIST *link)
```

***DESCRIPTION***

| | |
|---|---|
| src_im - | Source image with labeled objects |
| dst_im - | Destination image |
| link - | Link pointing to object |

copy_object() copies the labeled object from the source image to the same coordinate in the destination image.

***EXAMPLE***

```
To copy objects pointed at with the mouse to another image:

#include "image.h "
#include "im_aio.h"
LIST *l, *o;

readfile("cermet",a,0,0);
threshold(a,b,128);
invert_im(b,b);
l = list_label(b,c,8,0);
set_im_type(d, LABEL_2D);
while (o = point_object(c,l))
      { copy_object(c,d,o); display_image(d); }
l = rm_list(l);
```

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

g_copy_object  g_copy_object_to

## *copy_part_image*

## *copy_masked_part*

***NAME***

copy_part_image - copy a rectangular part of an image to another image

copy_masked_part - copy an arbitrary shaped part of an image to another image

***SYNOPSIS***

```
#include "im_proto.h"

int copy_part_image(IMAGE *in, IMAGE *out, int sx, int sy, int sz,
int width, int height, int depth, int dx, int dy, int dz)

int copy_masked_part(IMAGE *in, BOOL_MASK *mask, IMAGE *out, int sx,
int sy, int sz, int width, int height, int depth, int dx, int dy, int
dz, int clear)
```

***DESCRIPTION***

copy_part_image copies a part with dimensions "width" * "height" * "depth" from source position ("sx","sy","sz") in image "in" to the destination position ("dx","dy","dz") in image "out"

copy_masked_part uses a Boolean mask "mask" to copy a part with dimensions "width"*"height"*"depth" from source position ("sx","sy","sz") in image "in" to the destination position ("dx","dy","dz") in image "out". The bits which are set in the mask indicate which pixels from the rectangle of the source image must be copied to the destination image. The flag "clear" is added to indicate what to do with the pixels in the rectangle of the destination that are not in the Boolean mask. If "clear" is set to 0, the pixels are left untouched, otherwise they are cleared.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

copy_im  warp_image

## *covariance*

***NAME***

covariance - calculate the covariance between two images

***SYNOPSIS***

```
#include "im_proto.h"

double covariance(IMAGE *in1, IMAGE *in2)
```

***DESCRIPTION***

Calculate the covariance between the two images "in1" and "in2" and returns the result. The covariance is defined as:

```
      covar(a,b) = MEAN(in1*in2) - MEAN(in1)*MEAN(in2)
```

***RETURN VALUES***

The covariance.

---

## *covmatrix*

***NAME***

covmatrix - calculate covariance matrix of an image

***SYNOPSIS***

```
#include "im_proto.h"

int covmatrix(IMAGE *in, VAR_OBJECT *out, int width, int height)
```

***DESCRIPTION***

Calculate the covariance matrix of image "in" for window size "width" by "height" and stores the result in var_object "out".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

eigenvectors  eigenfilters

---

## *covplanematrix*

***NAME***

covplanematrix - calculate covariance matrix of an image

***SYNOPSIS***

```
#include "im_proto.h"

int covplanematrix(IMAGE *in, VAR_OBJECT *out)
```

***DESCRIPTION***

Calculate the covariance matrix between the planes of 3D image "in" and stores the result in var_object "out".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

im_eigenvectors  im_principle_component  covariance

---

## *cr*

***NAME***

cr - obtain contour ratio of object

***SYNOPSIS***

```
#include "im_aio.h"

double cr(LIST *link)
```

***DESCRIPTION***

| | | |
|---|---|---|
| link | - | Link pointing to object |

AIO primitive to obtain value of an object feature

cr() returns the contour ratio of the object pointed to by "link" if this has previously been measured.

***RETURN VALUES***

contour ratio of object on succes
0.0 if link is not an object or if contour ratio has not been measured

***SEE ALSO***

measure  object_shape_meas  object_dens_meas

---

## *creat*

***NAME***

creat - create a new file

***SYNOPSIS***

```
int creat(char *name, int mode)
```

***DESCRIPTION***

This function is an interface to the standard C function as implemented on the current system. The functionality of this function is:

creat() creates a new file or prepares to rewrite an existing file called "name", given as the address of a null-terminated string. If the file did not exist, it is given mode "mode", as modified by the process's mode mask (see umask(2)). Also see chmod(2) for the construction of the mode argument.

If the file did exist, its mode and owner remain unchanged but it is truncated to 0 length.

The file is opened for writing only (not reading), and its file descriptor is returned.

The mode given is arbitrary; it needs not allow writing. This feature is used by programs which deal with temporary files of fixed names. The creation is done with a mode that forbids writing. Then if a second instance of the program attempts a creat, an error is returned and the program knows that the name is unusable for the moment.

The system scheduling algorithm does not make this a true uninterruptible operation, and a race condition may develop if creat is done at precisely the same time by two different processes.

***RETURN VALUES***

-1 is returned if:
a needed directory is not searchable;
the file does not exist and the directory in which it is to be created is not writable;
the file does exist and is unwritable;
the file is a directory;
there are already too many files open.

***SEE ALSO***

write close open

## *create_clut*

***NAME***

create_clut - create a color lookup table

***SYNOPSIS***

```
#include "im_infra.h"

CLUT *create_clut(int type, char *name)
```

***DESCRIPTION***

Creates a color lookup table (clut) that can be attached to an image. "name" is the name that can be assigned to the table.

In the arrays "r", "g" and "b" of a CLUT (see below), RGB triplets for 256 colors can be specified. The "table" array is intended to be used by the user-interface to store the colormap entry that best represents the RGB triplet. Image does not fill or interpret these "table" values in any way.

The newly created table can be "prefilled" with values of often used lookup tables. The "type" parameter specifies filling method, possible values are:

| | |
|---|---|
| EMPTY_LUT_T | r, g and b filled with nulls |
| BLUE_LUT_T | b filled linear from 0 to 255 |
| GREEN_LUT_T | g filled linear from 0 to 255 |
| CYAN_LUT_T | b and g filled linear from 0 to 255 |
| RED_LUT_T | r filled linear from 0 to 255 |
| MAGENTA_LUT_T | r and b filled linear from 0 to 255 |
| YELLOW_LUT_T | r and g filled linear from 0 to 255 |
| GREY_LUT_T | r, g and b filled linear from 0 to 255 |
| LABEL_LUT_T | simulates the label display |
| MULTI_LUT_T | the three primary colors are turned on for each entry number that has a specified bit set (see set_rgb_bits() ) |
| OVERLAY_LUT_1_T | entries that have bit 1 set are displayed in red |
| OVERLAY_LUT_2_T | like previous but now bit 2 |
| .. | .. |
| OVERLAY_LUT_8_T | like previous but now bit 8 |

When a clut is created, it will be entered in a list for accounting. So always create a clut using this routine.

***STRUCTURES***

```
A clut is defined by the following structure:

typedef struct clut_t {
      char          clut_name[IM_NAMELEN];
      unsigned char r[256];
      unsigned char g[256];
      unsigned char b[256];
      unsigned long table[256];
} CLUT;
```

***RETURN VALUES***

The pointer to the newly allocated clut structure on success.
NULL on failure.

***SEE ALSO***

set_clut  set_rgb_bits  destroy_clut

---

## *create_display*

***NAME***

create_display - create a display window for an image without one

***SYNOPSIS***

```
#include "disp_p.h"

int create_display(IMAGE *im, int xp, int yp, int xs, int ys)
```

***DESCRIPTION***

This functions creates a display window for image "im" at position "xp", "yp" with sizes "xs", "ys". Only if the image did not have a display window attached, a new window will be made. Image that are created using create_image() do not have a display attached and can be given one by this function.

Images created with the function make_image() already have a display window. The result of create_image() followed by create_display() is identical to the result of make_image().

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

create_image  make_image

---

## *create_histogram*

## *destroy_histogram*

## *histo_data*

## *copy_histogram*

***NAME***

create_histogram - create a histogram object

destroy_histogram - destroy a histogram object

histo_data - get the histogram data from an image

copy_histogram - copy the data of one histogram to another

***SYNOPSIS***

```
#include "im_infra.h"
#include "im_proto.h"

HISTOGRAM *create_histogram(char *name, int chans, int dims, int
dim1, int dim2, int dim3, int dim4, int dim5)

int destroy_histogram(HISTOGRAM *histo)

HISTOGRAM *histo_data(IMAGE *image, HISTOGRAM *histo, int bins,
double minrange, double maxrange)

HISTOGRAM *copy_histogram(HISTOGRAM *srchisto, HISTOGRAM *desthisto)
```

***DESCRIPTION***

The histogram objects are capable of describing and containing multidimensional histograms. Along with the actual histogram data, the median value of the lowest and highest bin as well as the width of the bins can be stored. Thus fully describing the histogram data.

create_histogram() creates an empty histogram object. The string "name" is the name of the histogram by which it is listed in the dialog boxes. "chans", "dims" and "dim1" to "dim5" determine the sizes of the histogram. "chans" being the number of channels, "dims" the number of dimensions of the histogram and "dim1" to "dim5" the sizes of each dimension.

destroy_histogram() destroys the histogram object pointed to by the pointer "histo".

histo_data() calculates the histogram of image "image" and stores it in the histogram object "histo". If "histo" is NULL, a new histogram object is created with the name "Histogram_of_..." (... being the name of the image). The number of bins used for the histogram is determined by "bins". "bins" is 0 means that the routine will choose a value for the number of bins. For the integer based image types grey, binary and color, this value will be equal to the range of the image data. For the floating point image type float, it is set at 256. "minrange" and "maxrange" determine the range of the image data to be put in the histogram. If "minrange" is equal to "maxrange", the actual range of the image data will be taken.

copy_histogram() copies the data of histogram "srchisto" to the histogram "desthisto" thereby adjusting the sizes of "desthisto" to match those of "srchisto".

***EXAMPLE***

```
histo_data A New 0 128.0 254.0
```

will calculate the histogram for the data in image "A" that lies between 128 and 254, the histogram will be 127 bins (254-128 + 1) in the case of image "A" being an integer typed image.

```
histo_data A "histo1" 256 0.0 0.0
```

will calculate the histogram of all pixels in image "A" using 256 bins, the resulting histogram is named "histo1".

***STRUCTURES***

A histogram is defined by the following structure:

```
typedef struct histogram_t {
      void *publish;
      unsigned long *hdata;
      unsigned int nr_chans;
      unsigned int nr_dims;
      unsigned int dims[V_O_MAX_DIM];
      char name[IM_NAMELEN];
      char *comment;
      double lbin_median[V_O_MAX_DIM];
      double hbin_median[V_O_MAX_DIM];
      double bin_width[V_O_MAX_DIM];
      void *fut1;
      void *fut2;
      void *fut3;
      void *fut4;
} HISTOGRAM;
```

`V_O_MAXDIM` is defined in image.h (currently set at 5)

***RETURN VALUES***

| | |
|---|---|
| create_histogram: | pointer to the histogram or<br>NULL if creation failed. |
| destroy_histogram: | IE_OK (1) on destruction or<br>IE_NOT_OK (0) if the pointer was not a histogram. |

***SEE ALSO***

histogram_to_image histogram_to_var_object image_to_histogram
histogram_by_name histogram_ok is_histogram
histogram_comment dump_histogram list_histograms show_histogram_info

---

## *create_image*

***NAME***

create_image - create an image without a display window

***SYNOPSIS***

```
#include "im_proto.h"

IMAGE *create_image(char *name, int type, int lenx, int leny, int
lenz)
```

***DESCRIPTION***

create_image() creates an image of the specified type, with the specified name and of the specified dimensions. If "name" is equal to the name of an already existing image, that existing image is first destroyed.

***RETURN VALUES***

A pointer to the new image or
NULL on failure

***SEE ALSO***

destroy_image

---

## *create_live_window*

## *create_diff_window*

***NAME***

create_live_window - view/grab images from a framegrabber

create_diff_window - view/grab images from a framegrabber using a reference image

***PLATFORM***

MS-Windows.

***SYNOPSIS***

```
#include "image.h"

void create_live_window(IMAGE *image)

void create_diff_window(IMAGE *image)
```

***DESCRIPTION***

create_live_window() (=LiveGrabber in the menu) is an interactive tools that allows to view "live-video" from the framegrabber and grab images. If an image is grabbed, it is stored in the image "image". The tool offers several buttons and slider to control zooming and panning and some grabber settings.

The create_diff_window() is almost identical to the create_live_window() tool. It additionally offers the possibility to first grab a reference image and then view the video from the grabber as the difference between the actual image and that reference image.

***RETURN VALUES***

None

***SEE ALSO***

fg_grab_image

## *create_silo*

***NAME***

create_silo - create an image-silo

***SYNOPSIS***

```
#include "silo.h"

SILOPTR create_silo(char *siloname)
```

***DESCRIPTION***

| | | |
|---|---|---|
| siloname | - | filename for the image-silo. |

Creates a file to hold an image-silo. Its return value is a handle that must be passed to all silo-I/O routines.

Atfer creating the silo, it writes "MAXRECORD" (see silo.h) empty entries to this file to initialize the silo. Finally it calls open_silo() to open this file.

***RETURN VALUES***

Pointer to the newly created silo.
NULL if an error has occurred.

---

## *cst*

***NAME***

cst

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See contrast_stretch

---

## *cube_view*

***NAME***

cube_view - interactive viewer on binary 3d images

***SYNOPSIS***

```
#include "im2scil.h"

int cube_view(IMAGE *in, IMAGE *out, int plane, int view, int
interaction)
```

***DESCRIPTION***

Displays a binary 3D image in a pseudo-three-dimensional view on a bitplane using a simple back-to-front algorithm using a 2D grey value image to approximate a voxel.

The image is filled with 15 pixels to approximate 1 voxel in this form:

```
     ___ ___ ___
    | u | u | u |
 ___|___|___|___|
| f | f | f | s |
|___|___|___|___|
| f | f | f | s |
|___|___|___|___|
| f | f | f | s |
|___|___|___|___|
```

Where u is the upper side, f is the front and s is the right side of the voxel. parameters:

*in - binary 3D image
*out - output image
plane - plane number to view
view - viewing direction having value (1,2 or 3).

1 is front view looking in z direction at xy planes
2 is view on the right side of the image in x direction at zy planes.
3 is view to the top side of the image in y direction at xz planes.

interaction - 1 is yes, 0 is no

if interaction is yes, one may step through the back-to-front display of the planes by means of pointing in the upper/lower part of the image and pushing the right button, or using the keyboard pressing "f" for forward of "b" for backward. Pressing return ends the interaction mode.

The display rendering starts at the highest "plane" value and builds up with back-to-front displaying of planes until the entered plane number to view.

***RETURN VALUES***

None

## *decrement_im*

***NAME***

decrement_im - decrement image pixels

***SYNOPSIS***

```
#include "im_proto.h"

int decrement_im(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

Decrement each element of image "in" and store the result in the corresponding element of image "out".

***NOTE***

For more powerful image arithmetic expressions (scaling, adding offsets, etc), use the function eval().

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

increment_im  eval

## *default_images*

***NAME***

default_images - make the four images A, B, C and D

***SYNOPSIS***

```
#include "im2scil.h"

void default_images(int number)
```

***DESCRIPTION***

The images A, B, C and D, are created by this function at start-up. Also the two default var_objects "obj1" and "obj2" are created by this function. The function is called from the initialization file "scilinit". The position of the four images can be influenced by the set_start_pos() (also in the scilinit file). set_start_pos() must be called before the function "default_images".

The parameter "number" (max. value is 4), specifies the number of images to create.

When the call to default_images() is removed from the "scilinit" file no images and var_objects will be created at start-up.

This function must be called after "initimage"

***RETURN VALUES***

None

***SEE ALSO***

set_start_pos

## *defuz*

## *bernsen_threshold*

***NAME***

defuz - sharpening filter for grey value images using grey value morphology

bernsen_threshold - thresholding using grey value morphology

***SYNOPSIS***

```
#include "im_proto.h"

int defuz(IMAGE *in, IMAGE *out, int filt_x, int filt_y, double thr)

int bernsen_threshold(IMAGE *in, IMAGE *out, int filt_x, int filt_y,
int max_diff)
```

***DESCRIPTION***

defuz() performs a sharpening operation on the grey value image "in" and stores the result in the grey value image "out". The image is scanned with a moving window with sizes "filt_x" and "filt_y". For each position, both minimum and maximum value are determined. The values of the minimum, maximum and center pixel value are denoted by MIN, MAX and C resp. The value of the last parameter "thr" is denoted by THR. The new value of the center pixel is calculated as follows:

```
if(C < MIN + THR * (MAX - MIN))
   C = MIN;
else
   C = MAX;
```

The last parameter THR indicates a bias towards either the local minimum or the local maximum value. In case THR == 0.5 there is no bias towards a direction. This means that if THR == 0.5 the center value is substituted by either min or max depending on whichever of the two is closest in value. If THR has value 0 the result is identical to a local maximum operation and if THR has value 1 the result is that of a local minimum filter. For non-linear sharpening the value of 0.5 is recommended for THR.

The bernsen_threshold() operation is comparable to defuz() without bias. However, instead of replacing the value of the center pixel by the local minimum or maximum, this pixel is assigned 0 or 1. The result of this operation will become very noisy in areas where there is no distinct difference between minimum and maximum. Therefore, with the last parameter "max_diff" the user can indicate the minimum required difference between the local minimum and maximum. If the difference is less, the center pixel is assigned 1 by default.

***LITERATURE***

H.P. Kramer and J.B. Bruckner, Iterations of a non-linear transformation for enhancement of digital images, Pattern Recognition, vol.7, 1975, 53-58

J. Bernsen, Dynamic thresholding of grey-level images, 8th IAPR International Conference on Pattern Recognition (Paris), 1986, 1251-1255

***RETURN VALUES***

OK (1) on success
NOT_OK (0) on failure (see im_error.h)

---

## *dens*

***NAME***

dens

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See density

---

## dens_limits

**NAME**

dens_limits - set the limits for optical densitometric measurement of AIO

**SYNOPSIS**

```
int dens_limits(int minimum, int maximum, double max_opt_dens)
```

**DESCRIPTION**

dens_limits() can be used to set three global definition used in object densitometric measurement (object_dens_meas()) of the AIO package. "minimum", "maximum" and "max_opt_dens" are the variables that can be set. The relation between those variables and densitometric measurement is explained below. Legal values for "minimum" and "maximum" range from 0 to 32767, also "maximum" must be greater than "minimum". "max_opt_dens" can range from 0.0 to 4.5.

Transmission

Transmission (T) is defined as the proportion of incident light which passes through absorbent material, usually expressed as a percentage.

```
     I
T = ---- * 100%
     Io
```

where T is the transmission, "Io" is the incident light and "I" is the transmitted light, see also figure. Sometimes it is used as transmittance in which case it is represented as a fraction.

```
               /^\ I
                |  (transmitted light)
                |
        +================+
-----------| specimen |----------
        +================+
               /^\
                | Io
                | (incident light)
```

Light is absorbed progressively as it passes through a partly absorbent substance, if a single thickness reduces light intensity to half its original value, a double thickness will not absorb all light, but will reduce it to 25 percent.

Optical Density

Optical density (O.D.) is defined as the common base logarithm of the ratio of incident to transmitted light:

```
O.D. = - log10(Transmittance)
```

In theory the optical density ranges from 0 (at 100% transmission) until infinity (for zero transmittance or completely opaque objects). In a practical setup infinity will not be reached. The densitometric  measurements require calibration that means two intensity values  (minimum, maximum) are mapped on the lower and upper limit of the  optical density according to a table. As the intensity ranges from  "min_intensity" to "max_intensity" the optical density is calculated  according:

```
                                          -MAXOD
          (I - min_intensity) * (1.0 - 10.0      )       -MAXOD
 -log10( --------------------------------------- + 10      )
          (max_intensity - min_intensity)
```

This formula reaches MAXOD at I = min_intensity and equals to zero  if I = max_intensity, it has a perfect logarithmic behavior. The default for MAXOD is 2.55, which is a sensible value for camera  systems where a dynamic range from 0 to 255 is expected limiting  the optical density in ideal circumstances to 2.40. In the case of having 16 useful bits the upperlimit of optical density is 4.5.

The minimum intensity value and the maximum intensity values may  be used to calibrate the measurements. For instance this is useful to correct for glare or straylight originating from the optical system.

***LITERATURE***

TV Microscopical Image Analysis for Accurate DNA Quantification in Pathology. Ph.D. Thesis T.K. ten Kate

***RETURN VALUES***

IE_OK (1) on success
IE_NOT_OK (0) when values are out of range

***SEE ALSO***

object_dens_meas  measure

## *density*

### *NAME*

density - object size and density measurement

### *SYNOPSIS*

```
#include "im_proto.h"

int density(IMAGE *label_im, IMAGE *grey_im, char *fname, int append)
```

### *DESCRIPTION*

Measure objects in the image "grey_im" using the labeled image "label_im" as an object indicator and write these parameters in a table on the console or in a text file. For each object present int the image "label_im", the corresponding object pixels in the image "grey_im" are used for the measurement. The measured parameters are:

- the coordinates of the object’s center of gravity
- the total object density (the sum of the pixel values within the object)
- the object size in number of pixels
- ratio of the object density and the object size (average pixel value within the object).

The measured parameters are written to the text file "fname" if specified, or printed on the controlling terminal ("fname" = NULL). If "append" is set (=1) then the generated table will be appended to the text file "fname" (if specified and existing). If "fname" already exists and "append" is not set (=0) then the file "fname" will be overwritten.

### *RETURN VALUES*

IE_OK (1) on success
Negative error status on failure (see im_error.h)

### *SEE ALSO*

calibrated_density  label  shape  measure  list_label  aio_label

## *destroy_clut*

***NAME***

destroy_clut - destroy a color lookup table

***SYNOPSIS***

```
#include "im_infra.h"

int destroy_clut(CLUT *clut)
```

***DESCRIPTION***

The specified clut is thrown away after checking that it is not one of the standard cluts that were defined at start up. These can not be thrown away by this function.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

create_clut

---

## *destroy_image*

***NAME***

destroy_image - destroy an image

***SYNOPSIS***

```
#include "im_proto.h"

int destroy_image(IMAGE *im)
```

***DESCRIPTION***

Destroy_image destroys the specified image. Also the possibly connected display window is destroyed

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

create_image

---

## *destroy_var_object*

***NAME***

destroy_var_object - destroy a var_object

***SYNOPSIS***

```
#include "objectsp.h"

int destroy_var_object(VAR_OBJECT *obj)
```

***DESCRIPTION***

"destroy_var_object" destroys the var_object specified by the pointer "obj".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

var_object  var_object_by_name  show_var_object_info

## *dialog_options*

***NAME***

dialog_options - change behaviour of the dialog generator.

***SYNOPSIS***

```
#include "md_gen.h"

int dialog_options(int nfew, int nsome, int nmany, int nhuge, int
f_feedb, int s_feedb, int m_feedb, int h_feedb, int g_feedb, int
with_arrows, int with_range)
```

***DESCRIPTION***

dialog_options() can be used to change the form of the dialog box generated by the dialog generator. "nfew", "nsome", "nmany" and "nhuge" define the limits of the ranges for the representation of numbers and choices. The representation can be modified by the other arguments. "f_feedb" specifies the representation of choices when the number of choices is less than or equal to "nfew", "s_feedb" the range from "nfew" up to "nsome", "m_feedb", the range from "nsome" up to "nmany", "h_feedb" the range from "nmany" up to "nhuge" and "g_feedb" the range from "nhuge" upwards.

"f_feedb" and "s_feedb" can be one of MARKED (1), INVERTED (2) or CYCLE (3)
"m_feedb" can be either INVERTED (2) or CYCLE (3)
"h_feedb" can be one of SLIDER (4), NUMBER (5) or TEXT (6)
"g_feedb" can be either NUMBER (5) or TEXT (6)

- MARKED is represented by buttons with the value (numeric or symbolic) next to the button.
- INVERTED is represented by buttons with the value on the button and all buttons are shown at the same time.
- CYCLE is represented by buttons with the value on the button, but the buttons are shown one at the time.
- SLIDER is represented by a slider bar.
- NUMBER is represented by a number.
- TEXT is represented by text.

The method for presenting and changing values in each of the ranges may differ slighty per windowing system. Some windowing systems offer other (more convenient ways to represent a (range of) numbers/choices.

***RETURN VALUES***

None

## *dialog_stay_up*

***NAME***

dialog_stay_up - remove the dialog box automatically

***PLATFORM***

Unix.

***SYNOPSIS***

```
#include "md_gen.h"

int dialog_stay_up(int flag)
```

***DESCRIPTION***

The dialog box that is popped up if a menu item is selected, remains on screen until it is explicitly removed using the CANCEL button. This function is a switch to set this behaviour. If the dialog box is to be removed automatically, then "flag" must be "0" (zero). "flag" set to "1" will result in the dialog box stay on the screen

***RETURN VALUES***

None

---

## *dialog_wm*

***NAME***

dialog_wm - prevent dialog box movement in X-windows

***SYNOPSIS***

```
#include "md_gen.h"

int dialog_wm(int number)
```

***DESCRIPTION***

Since some X window managers decorate windows with stuff like titlebars, thereby displacing the dialog box each time it is mapped  to the screen, the dialog_wm function can be used to correct for  this behaviour. "number" is the number of pixels used for the correction of the vertical displacement

***EXAMPLE***

An example with the TWM window manager. To correct for the 20 pixel displacement place the following in the scilinit file:

```
dialog_wm(20);
```

***RETURN VALUES***

None

---

## *different_ok*

***NAME***

different_ok - check if two values differ from each other

***SYNOPSIS***

```
#include "im_infra.h"

int different_ok(int val1, int val2, char *text)
```

***DESCRIPTION***

The values "val1" and "val2" are matched and if they are equal an error is generated and the following message is added to the error-stack:

```
      <text> [<val2>,<val2>] must be different
```

***RETURN VALUES***

IE_OK (1) if the values differ
IE_NOT_OK (0) if the values are equal

## *dilation3x3*

***NAME***

dilation3x3 - dilation

***SYNOPSIS***

```
#include "im_proto.h"

int dilation3x3(IMAGE *in, IMAGE *out, int iter, int con, int bound)
```

***DESCRIPTION***

Performs a dilation ("expansion") on image "in" and stores the result in image "out". For each pixel in "in" a 3*3 neighborhood is scanned for non-object pixels (pixels with value 0). If the central pixel is a non-object pixel and at least one of the pixels in the neighborhood is an object pixel, the central pixel becomes an object pixel (value 1).
The exact definition of the neighborhood depends upon the connectivity argument "con". Connectivity can be 4 or 8 connected, but also 48 or 84 can be specified, allowing for alternating connectivity on each iteration to approach a 6 connected neighborhood. With the "bound" argument the pixels around the edge of the image are either taken as 1 or 0.

The effect of this operation is expansion of the objects with (an) extra contour(s). The number of contours added depends upon "iter".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

erosion3x3  arbit_dilation arbit_erosion

## *dir_maximum*

### *NAME*

dir_maximum - view maximum pixel value along an axis

### *SYNOPSIS*

```
#include "im_proto.h"

int dir_maximum(IMAGE *in, IMAGE *out, int dir)
```

### *DESCRIPTION*

dir_maximum() calculates the maximum pixel value on each scanline along an axis of the 3D image "in" and stores these in the 2D image "out". The axis on which the function operates is determined by "dir" (X = 1, Y = 2, Z = 3).

### *RETURN VALUES*

IE_OK (1) on success
Negative error status on failure (see im_error.h)

### *SEE ALSO*

sfp  z_planes  pix_maxval

## *disp_circle*

## *disp_draw_mode*

## *disp_draw_value*

## *disp_oval*

## *disp_rect*

## *disp_srect*

## *disp_text*

## *disp_text_font*

## *disp_vector*

### *NAME*

disp_circle - draw a circle

disp_draw_mode - set the drawing mode

disp_draw_value - set the drawing value

disp_oval - draw an oval

disp_rect - draw a rectangle

disp_srect - draw a filled rectangle

disp_text - draw some text

disp_text_font - set text font

disp_vector - draw a line

### *SYNOPSIS*

```
#include "disp_p.h"

int disp_circle(IMAGE *im, int x, int y, int r)

int disp_draw_mode(int mode)

int disp_draw_value(int value)

int disp_oval(IMAGE *im, int x, int y, int xr, int yr)

int disp_rect(IMAGE *im, int x, int y, int xsize, int ysize)

int disp_srect(IMAGE *im, int x, int y, int xsize, int ysize)

int disp_text(IMAGE *im, int x, int y, char *str)
int disp_text_font(char *font)

int disp_vector(IMAGE *im, int x1, int y1, int x2, int y2)
```

***DESCRIPTION***

Graphical routines which effect the display of images only.

These routines can be used to draw graphical primitives in the display window of an image. The image itself is not altered by the drawing operation. The origin (0,0) of the specified coordinate system is the upper left corner of the image. The positive x-axis is going to the right and the positive y-axis is going down.

disp_circle draws a circle of radius "r" at coordinates ("x","y") in the specified image.

disp_draw_mode sets the drawing mode:

| | |
|---|---|
| 1 | is the copy-mode |
| 2 | is the or-mode |
| 3 | is the xor-mode. |

disp_draw_value sets the color to be used when drawing.

disp_oval draws an oval with the specified radius in X-direction "rx" and radius in Y-direction "ry" at ("x","y") in the specified image.

disp_rect draws a rectangle of "xsize"*"ysize" at ("x","y") in the specified image.

disp_srect draws a filled rectangle of "xsize"*"ysize" at ("x","y") in the specified image.

disp_text draws string "str" at ("x","y") in the specified image.

disp_text_font determines which font is to be used when writing to the display of an image with disp_text

disp_vector draws a straight line from ("x1","y1") to ("x2","y2") in the specified image.

***RETURN VALUES***

IE_OK (1)

## *display_image*

***NAME***

display_image - display an image

***SYNOPSIS***

```
#include "im2scil.h"

int display_image(IMAGE *im)
```

***DESCRIPTION***

display_image() displays the specified image "im", provided the image has a display window attached, and the display mode is on. The display mode can be enabled/disabled with auto_display().

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

part_image_display  make_image  destroy_image  auto_display  set_display_mode
set_display_slice  next_plane  auto_plane

## *dist_average*

***NAME***

dist_average - texture measure, average of distance transform

***SYNOPSIS***

```
#include "im_proto.h"

double dist_average(IMAGE *input, IMAGE *mask, double threshold, int
background)
```

***DESCRIPTION***

Threshold the image, do distance transform and average the image.

If "background" is nonzero, the thresholded image is inverted before doing the distance transform.

The averaging of the image is only done in the areas where the bit-image "mask" has value 1.

***RETURN VALUES***

The texture value is returned. In case of error, this is 0.

***SEE ALSO***

box_dimension  gld_mean  gld_entropy gld_contrast  gld_asymmetry
glc_entropy  glc_contrast  glc_asymmetry  glr_nonuniformity
glr_shortrunemphasis  glr_longrunemphasis  glr_greynonuniformity
glr_percentage  edge_average  dist_average

## *dist_skelet*

***NAME***

dist_skelet - skeleton based on distance transform

***SYNOPSIS***

```
#include "im_proto.h"

int dist_skelet(IMAGE *in, IMAGE *out, int angle, int hstep, int
dstep, int action)
```

***DESCRIPTION***

Calculate a skeleton from the distance transformed image "in" and store the resulting skeleton in the binary image "out". The skeleton is defined as a set of connected, one pixel thick arcs, lying midway between the object boundaries and being a topological retraction with the same connectivity as the original object. The skeleton represents the morphologic ("shape") features of the original object.

"angle" defines the sensitivity of the skeleton for small bulges in the object contour causing the algorithm to create branches. It specifies the maximum angle for circumscribing arcs of a bulge which will generate a branch in the skeleton. A small value allows only sharp corners in the object's contour to generate branches. A large value specifies that even smooth corners may generate branches. "angle" is specified in units of 45 degrees. "hstep" and "dstep" are the step-sizes that were used to generate the distance transformation image.

The parameter "action" defines how far the skeleton is processed. The value "interact_points" (0) specifies that only the medial axis is to be calculated, this skeleton may not be connected. The  value "thick" (1) specifies that a connected skeleton is to be generated from the medial axis, this skeleton can be more than one pixel thick at places. The value "thinned" (2) also performs thinning of the completely connected skeleton to a one pixel thick skeleton.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

distance  holt_skelet  hild_skelet

## *distance*

***NAME***

distance - distance transformation

***SYNOPSIS***

```
#include "im_proto.h"

int distance(IMAGE *in, IMAGE *out, int hstep, int dstep, int kstep,
int edge)
```

***DESCRIPTION***

Applies a distance transformation to the binary image "in" and stores the resulting distance values in the corresponding pixels of the grey valued image "out". A distance transformation replaces each pixel of an object with an estimate of its shortest distance to the background (the distance to the nearest background pixel). The way the distance between pixels is defined is specified by "hstep", "dstep" and "kstep". "hstep" defines the distance between two pixels that are horizontally or vertically connected. "dstep" defines the distance between pixels that are diagonally connected. "kstep" is the distance between two pixels that are a "knight’s move" apart. "edge" specifies if the objects extends beyond the border of the image (On=1) or not (Off=0).

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

dist_skelet

## *dither*

***NAME***

dither - graphic dotting of a grey valued image

***SYNOPSIS***

```
#include "im_proto.h"

int dither(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

Replace the pixels in a grey_2d image "in" by a black (0) and white (255) dot pattern, with a density equivalent to the original grey value and store the result in image "out". The effect of this operation is a pseudo-grey value image.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

pseudo  greduce

## do_alert

### NAME

do_alert - system independent alert routine

### SYNOPSIS

```
#include "md_gen.h"

int do_alert(char *str1, ...)
```

### DESCRIPTION

do_alert is the system independent alert routine of scil.

do_alert generates an alert box with messages and buttons. It forces the user to respond by selecting one of the buttons, since all other program activity is stopped.

do_alert can be called with up to 15 strings of messages, each of which are printed on a separate line. Any message line may also contain newline '\n' characters.

The last string argument to do_alert must be a button specification string of the form "[BUT1]...[BUTn]". The maximum number of buttons is 8.

do_alert returns with the selected button number. Buttons are numbered 1 .. 8.

### NOTE

In the MS-Windows version a more versatile alert mechanism is provided through the function typed_alert().

### EXAMPLE

```
int choice;
choice = do_alert("This is an alert\n\n WOW\n\n",
      "Nice isn't it", "[Yes it is][No it is not]");
if(choice == 1)
      printf("I agree\n");
else
      printf("I don't think I like you very much\n");
```

### RETURN VALUES

The number of the selected button. Buttons are numbered 1 .. 8.

## *dots*

***NAME***

dots

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See dither

---

## *draw_line*

***NAME***

draw_line - draw a line in a image

***SYNOPSIS***

```
#include "im_proto.h"

int draw_line(IMAGE *image, int x1, int y1, int x2, int y2, int
value)
```

***DESCRIPTION***

Draw a straight line segment of value "value" into the image "image"  from the pixel with coordinates "x1","y1" to the pixel with  coordinates "x2","y2".

When drawing more than one line, it is recommended to turn of the automatic display of the images (see auto_display()), draw the lines and turn on the display of the images again and display the image to view the result.

***RETURN VALUES***

IE_OK (1) on success
IE_NOT_OK (0) on failure

---

## *drawcurve*

***NAME***

drawcurve - draw a piece-wise linear curve

***SYNOPSIS***

```
#include "grey_2dp.h"

int drawcurve(VAR_OBJECT *input, IMAGE *output, int value, int
smooth, int circ)
```

***DESCRIPTION***

A piece-wise linear curve is drawn, based on the series of coordinate pairs specified in the 2-dimensional var_object "input". Each row of "input" contains the coordinates of one point. A number of adjacent points, specified by the value of "smooth", are averaged and the mean value replaces the considered point. (That is, internally; the data in "input" are not changed). The coordinates of the smoothed set are then converted to the nearest integral pixel position and the points successively connected by straight lines. The obtained curve is drawn into the image "output" using "value" as drawing value.

If "circ" is Yes (1), the pixel corresponding with the last coordinate pair in "input" will be connected to the pixel corresponding with the first coordinate pair in "input". Also for the smoothing operation these points are considered as adjacent.

***RETURN VALUES***

IE_OK (1) on success
IE_NOT_OK (0) on failure

***SEE ALSO***

maximum_trace maximum_cost_path resample_perp back_project

## *dump_var_object*

***NAME***

dump_var_object - show all data of a var_object (in ASCII)

***SYNOPSIS***

```
#include "objectsp.h"

int dump_var_object(VAR_OBJECT *object, char *filename, int number)
```

***DESCRIPTION***

The data of var_object "object" is dumped in ASCII to either the terminal or a file. If a name is specified for the file ("filename") then the data will be stored in a file, else, if "filename" is a NULL pointer or an empty string, the data will be dumped on the terminal. The last parameter "number" specifies the number of values that will be printed on a single line (default = 1).

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

var_object  read_var_object  write_var_object

## *dyn_link*

## *dyn_unlink*

***NAME***

dyn_link - add a shared library at runtime

dyn_unlink - remove a shared library at runtime

***PLATFORM***

UNIX (Sun, SGI, HP)

***SYNOPSIS***

```
int dyn_link(char *library, int verbose)

int dyn_unlink(char *library, int verbose)
```

***DESCRIPTION***

dyn_link() adds the shared library file "library" runtime to SCIL_Image. This adds functionality while the program is running. dyn_unlink() removes the library file "library" again from the running program. If the "verbose" flag is on (1), the names of the functions that are being linked/unlinked are listed. If "verbose" is off (0) nothing is listed.

In SCIL_Image, the number of libraries that can be linked dynamically is limited to 20 and the maximum number of functions/variables that can be accessed from each library is 128.

To link a shared library to SCIL_Image you must have two files with the same name in one directory, the shared library itself (".so" extension) and a Command Description File (".cdf" extension). In the CDF-file the functions in the library are described in SCIL_Image format. (See chapters 3 & 5 of the User Manual). Without a CDF-file, a shared library cannot be linked into SCIL_Image.

SCIL_Image will search for these files first in the current directory and if it does not find them there, it will search in the directories specified in the environment variable SCIL_DYN.

Creating a shared library is dependent upon the platform and operating system used. For a detailed description, read the manual pages of the C compiler (cc) and the linker (ld) on your platform.

***RETURN VALUES***

OK (1) on success
NOT_OK (0) on failure

## *eccentr*

***NAME***

eccentr - obtain eccentricity of object

***SYNOPSIS***

```
#include "im_aio.h"

double eccentr(LIST *link)
```

***DESCRIPTION***

| | | |
|---|---|---|
| link | - | Link pointing to object |

AIO primitive to obtain value of an object feature

eccentr returns the eccentricity of the object pointed to by "link" if this has previously been measured.

***RETURN VALUES***

eccentricity of object on success
0.0 if link is not an object or if eccentricity has not been measured

***SEE ALSO***

measure  object_shape_meas  object_dens_meas

## *ecvt*

## *fcvt*

## *gcvt*

***NAME***

ecvt, fcvt, gcvt - output conversion

***SYNOPSIS***

```
char *ecvt(double value, int ndigit, int *decpt, int *sign)

char *fcvt(double value, int ndigit, int *decpt, int *sign)

char *gcvt(double value, int ndigit, char *buf)
```

***DESCRIPTION***

These functions are interface functions to the standard C functions as implemented on the current operating system. The functionality of these functions is:

ecvt() converts "value" to a null-terminated string of ndigit ASCII digits and returns a pointer thereto. The position of the decimal point relative to the beginning of the string is stored indirectly through "decpt" (negative means to the left of the returned digits). If the sign of the result is negative, the word pointed to by "sign" is non-zero, otherwise it is zero. The low-order digit is rounded.

fcvt() is identical to ecvt(), except that the correct digit has been rounded.

gcvt() converts "value" to a null-terminated ASCII string in "buf" and returns a pointer to "buf". It attempts to produce "ndigit" significant digits in E format, ready for printing. Trailing zeros may be suppressed.

***SEE ALSO***

printf

***BUGS***

The return values point to static data whose content is overwritten by each call.

## *edge_average*

***NAME***

edge_average - texture measure, how much edge is in the image

***SYNOPSIS***

```
#include "im_proto.h"

double edge_average(IMAGE *input, IMAGE *mask, int filtersize, int
usegrad)
```

***DESCRIPTION***

Do a low-pass filter of the image "input", using a "filtersize"*"filtersize" uniform filter which is performed twice. Subtract that image from the original. In that image, the zero crossings are detected.

If "usegrad" is nonzero, the "input" image is filtered by a gradient filter, also using a "filtersize" kernel. The average of the gradient values AT THE ZERO CROSSINGS is the texture value.

If "usegrad" is zero, the number of skeleton segments in the zero crossings image is counted. This number, divided by the number of pixels in the zero crossing image, is used as the texture value.

The calculation of the texture is only done in the areas where the bit-image "mask" has value 1.

***RETURN VALUES***

The texture value is returned. In case of error, this is 0.

***SEE ALSO***

box_dimension  gld_mean  gld_entropy gld_contrast  gld_asymmetry
glc_entropy  glc_contrast  glc_asymmetry  glr_nonuniformity
glr_shortrunemphasis  glr_longrunemphasis  glr_greynonuniformity
glr_percentage  edge_average  dist_average
uniform  zcross

## *edge_object*

***NAME***

edge_object - check whether object touches edge of image

***SYNOPSIS***

```
#include "im_aio.h"

int edge_object(IMAGE *image, LIST *link)
```

***DESCRIPTION***

| | | |
|---|---|---|
| image | - | Pointer to image with objects |
| link | - | Link pointing to object |

edge_object() determines whether the object "link" touches the edges of the image "image".

***RETURN VALUES***

| | |
|---|---|
| 1 | if the object touches the edge |
| 0 | if it does not touch the edge |

Negative error status on failure (see im_error.h)

## *edge_ok*

***NAME***

edge_ok - check if the edge bit parameter is correct

***SYNOPSIS***

```
#include "im_infra.h"

int edge_ok(int bound)
```

***DESCRIPTION***

A lot of operations for binary images have a parameter to specify whether to set the border pixels or not, this function checks to see if that parameter is in the correct range of 0..1. If it is not an error is generated and the following message is added to the error-stack:

Edge bits parameter [<bound>] out of range (0..1).

***RETURN VALUES***

IE_OK (1) if "bound" is either "0" or "1"
IE_NOT_OK (0) if "bound" is any other value

***SEE ALSO***

range_ok

## *edge_preserve*

***NAME***

edge_preserve - edge preserving smoothing (general)

***SYNOPSIS***

```
#include "im_proto.h"

int edge_preserve(IMAGE *in1, IMAGE *in2, IMAGE *out, int filtx, int
filty)
```

***DESCRIPTION***

Combine a low pass filtered image "in1" with a variance filtered image "in2" to perform an edge preserving smoothing and store the result in image "out".
A low pass filter replaces each pixel of its input image by some average of the pixel values in a neighborhood. An example is the uniform filter.
A variance filter replaces each pixel of its input image by some estimate of the variance of the pixel values in a neighborhood. For example, a minimum filter subtracted from a maximum filter. To make sense, the low pass filtered image "in1" and the variance filtered image "in2" should derive from the same original image. The resulting image "out" is a smoothed, edges preserved, version of this original image.
The operation uses a window "filtx"*"filty". This same neighborhood should have been used in the filters producing image "in1" and "in2".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

kuwahara

## *edgps*

***NAME***

edgps

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See edge_preserve

## *eigen_segmentation*

***NAME***

eigen_segmentation - segments a textured image by eigenfilters

***SYNOPSIS***

```
#include "im_proto.h"

int eigen_segmentation(IMAGE *in, IMAGE *out, int size, int nr,
double scale)
```

***DESCRIPTION***

Performs a segmentation on a textured image by means of eigenfilters. The size of the square eigenfilters are given by "size". After calculation of "size"*"size" number of eigenfilters, the last "nr" ones are taken for convolution. Local energy calculation is performed (squaring the image and gaussian smoothing with "scale") and a non-linear transform is applied (ln (1+x)). From the resulting feature set the largest principle component is obtained by means of Karhunen-Loeve transform. The final result is stored in image "out".

***LITERATURE***

M. Unser and M. Eden, Nonlinear operators for improving texture segmentation based on features extracted by spatial filtering, IEEE Transactions on Systems, Man, and Cybernetics, vol. 20, 1990, 804-815.

***RETURN VALUES***

IE_OK (1) on succes
Negative error status on failure (see im_error.h)

***SEE ALSO***

eigenfilters  convolution  mul_im  vgauss  ln_im  karhunen_loeve

## *eigenfilters*

***NAME***

eigenfilters - calculate eigenfilters of an image

***SYNOPSIS***

```
#include "im_proto.h"

int eigenfilters(IMAGE *in, IMAGE *out, int width, int height)
```

***DESCRIPTION***

Performs an eigenvector analysis on the covariance matrix of "in" and stores the resulting vectors in de (3D) image "out". The covariance matrix is calculated for window size "width" by "height". One can segment a textured image by calculating the eigenfilters and applying for each filter (2D-plane of "out") convolution() and calculating the local energy (squaring the image and smoothing).

***EXAMPLE***

```
readf texttank
eigenfilters A B 3 3
apply_spatial_bank A B C 1 -1
mul_im C C C
vgauss C C 5.0 5.0 1.0 .99 .99 .99 -1 -1 1
sqrt_im C C
karhunen_loeve C D 0 0
```

***LITERATURE***

F. Ade, Characterization of textures by eigenfilters, Signal Processing, vol. 5, 1983, 451-457.

M. Unser, On the approximation of the discrete Karhunen-Loeve transform for stationary processes, Signal Processing, vol. 7, 1984, 231-249.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

covmatrix  eigen_segmentation

## *eigenvectors*

***NAME***

eigenvectors - calculate the eigenvectors and eigenvalues of an object

***SYNOPSIS***

```
#include "im_proto.h"

int eigenvectors(VAR_OBJECT *obj, VAR_OBJECT *vecs, VAR_OBJECT *vals)
```

***DESCRIPTION***

Performs an eigenvector analysis on the two-dimensional input "obj" and returns the eigenvectors in "vecs" (2D) and the eigenvalues in "val" (1D). If NULL is given for "vecs" or "vals", the corresponding data is not stored.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

covmatrix  covplanematrix

## *entropy_threshold*

***NAME***

entropy_threshold - thresholding using the entropy method.

***SYNOPSIS***

```
#include "im_proto.h"

int entropy_threshold(IMAGE *in, IMAGE *out, double fraction)
```

***DESCRIPTION***

Perform thresholding operation on the grey value image "in" and store the result in the binary image "out". The threshold-level is determined by the entropy method. This method is based upon an entropy measure in the grey level histogram of the image. The algorithm divides the histogram into two parts, minimizing the interdependence between the two parts, measured in terms of entropy.
The grey level that performs this division will be the threshold value.
As a condition, the user may specify the "fraction" of the image that minimally should be assigned to be a foreground object. The algorithm then searches for the minimal entropy within this constraint.

***RETURN VALUES***

The used threshold value.

***SEE ALSO***

threshold  isodata_threshold

## *eql*

***NAME***

eql

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See equalize

---

## *equal_images*

***NAME***

equal_images - test if contents of two images is equal

***SYNOPSIS***

```
#include <im_proto.h>

int equal_images(IMAGE *im1, IMAGE *im2)
```

***DESCRIPTION***

equal_images() tests if the images "im1" and "im2" are the same, that is, if the type and sizes of the images are same, and if so, if the contents of the images is equal. It returns TRUE (1) if the images are equal and FALSE (0) if they are not.

***RETURN VALUES***

TRUE (1) if the image contents is equal
FALSE (0) if the image contents is not equal
Negative error status on failure (see im_error.h)

---

## *equalize*

***NAME***

equalize - histogram equalization

***SYNOPSIS***

```
#include "im_proto.h"

int equalize(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

Change the values of the pixels in image "in" is such a way that the histogram is equalized, i.e. the histogram of "out" is as uniform (flat) as possible.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

clip  threshold  contrast_stretch tri_state_threshold lookup

---

## *equivalent_im*

***NAME***

equivalent_im - pixel equivalence of images

***SYNOPSIS***

```
#include "im_proto.h"

int equivalent_im(IMAGE *in1, IMAGE *in2, IMAGE *out)
```

***DESCRIPTION***

Set the value of a pixel in "out" according to the equivalence of the corresponding pixels in "in1" and "in2".

***NOTE***

For more powerful image arithmetic expressions (scaling, adding offsets, etc.), use the function eval .

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

---

## *eqv*

***NAME***

eqv

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See equivalent_im

---

## *erosion3x3*

***NAME***

erosion3x3 - erosion

***SYNOPSIS***

```
#include "im_proto.h"

int erosion3x3(IMAGE *in, IMAGE *out, int iter, int con, int bound)
```

***DESCRIPTION***

Performs an erosion ("shrinking") on the binary image "in" and stores the result in "out". For each pixel in "in" a 3*3 neighborhood is scanned for object pixels (pixels with value 1). If the central pixel is an object pixel and at least one of the pixels in the neighborhood is an non-object pixel, the central pixel is deleted as an object pixel (it becomes a background pixel, value 0). The exact definition of the neighborhood depends upon the connectivity argument "con".
Connectivity can be 4 or 8 connected, but also 48 or 84 can be specified, allowing for alternating connectivity on each iteration to approach a 6 connected neighborhood.
With the "bound" argument the pixels around the edge of the image are either taken as 1 or 0.

The effect of this operation is the deletion of the object contours. The number of contours deleted depends upon "iter".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

dilation3x3  arbit_erosion

---

## *err_report*

***NAME***

err_report - report on last error in silo package

***SYNOPSIS***

```
#include "silo.h"

void err_report(void)
```

***DESCRIPTION***

The silo package keeps track of errors in the variable "silo_err". This function gives a description of the last error that did occur. If no error has occurred, nothing will happen.

The error convention used in the silo package is as follows:

A positive (non zero) number is returned if no error occurred.

Zero is returned if a pointer error has occurred.

A negative number is returned on all other errors.

***RETURN VALUES***

None

## *eval*

***NAME***

eval - dynamic grey/float valued image expression evaluation

***SYNOPSIS***

```
#include "im_proto.h"

int eval(char *expression, int border)
```

***DESCRIPTION***

The command eval() enables the user to do simple operations on images, using a very intuitive syntax. The images may be either GREY_2D or FLOAT_2D or any combination of these two types. The type of the resulting image is the type the right-hand of the assignment evaluates to.

For example to add the images A and B pixel wise and store the result in C the command:

```
eval "C=A+B"
```

is all it takes. Addition of images is not limited to two input images, the command:

```
eval "D=A+B+C"
```

can also be used. Images can even be used more than once in the same command:

```
eval "A=(B+C)*B"
```

The operands in the eval() expression are not restricted to images, constants can also be used. The next command shows how to calculate the non-linear Laplace of the original image A.

```
grey_erosion A B 5 5
grey_dilation A C 5 5
eval "D=-B+2*A-C"
```

Constants can be of type integer (octal, decimal or hexadecimal) or floating point.

Example: (examples are given in a simplified notation)

```
octal             eval a=021
decimal           eval a=17
hexadecimal       eval a=0x11
floating point    eval a=17.0
                  eval a=1.7e1
```

Although the calculations done with eval() are slower than using compiled routines the advantages are obvious:

i) grey-valued intermediate results are calculated in 32 bit precision, only the final result is converted to 16 bits, and
ii) the simple syntax makes it very easy to use.

Not only pixel wise expressions are supported in the eval() command, it is also possible to define neighborhood operations using image indexes. The indexes can also be expressions. Expressions can be up to, plus or minus, the size of the image.

Example:
```
eval a=a[1+1,1-1]
eval a=a[256,0]         (valid if x > 256)
eval a=a[0,-256]        (valid if y > 256)
```

The following command calculates the Laplacian of image A and stores the result in B:

```
eval "B = 4 * A - A[1,0] - A[0,1] - A[-1,0] - A[0,-1]"
```

When using this feature do not expect great speed. Just experiment with it.

The second feature of eval() which is of great use is the availability of the x and y coordinates of the pixel which is processed. The next command makes a grey value ramp in image A:

```
eval "A=xx"
```

Note that the x coordinate is known under the name xx. The y coordinate is known under the name yy. To calculate an image A[x,y]=sqrt(x*x+y*y) use something like:

```
eval "A=sqrt((xx-128)*(xx-128) + (yy-128)*(yy-128))"
```

As can be seen from the previous example some mathematical functions are available in the eval() command. The following functions can be used:

```
abs(), cbrt(), exp(), hypot(), log(), pow(), sqrt() and rnd()
```

The third most important feature in eval() is the use of conditional expressions. To generate a circle with its center at position 128,128 and radius 100 use:

```
eval "A=(((xx-128)*(xx-128)+(yy-128)*(yy-128))>10000)?255:0"
```

Another example of the use of a conditional expression is given below where an image is thresholded at a level of 128:

```
eval "A= (B > 128) ? 255 : 0"
```

Also Boolean operators are allowed in the eval() command. To set a single point in the center of image A use:

```
eval "A = ((xx==128)&&(yy==128)) ? 255 : 0"
```

Parser

The parser accepts the operators listed below, they are standard C except for the "&+" (max) and "&-" (min) operators. It uses the priorities and associativities as defined in "The C programming language" by Kernighan and Ritchie(pages 214-215).

The associativity and precedence of the "&+" and "&-" operators are chosen to be those of the "+" and "-", this is an arbitrary choice, other choices may be "better".

| OPERATORS | ASSOCIATIVITY | PRECEDENCE |
|---|---|---|
| () [] | left-associative | HIGHEST |
| - ! ~ | right_associative | |
| * / % | left_associative | |
| + - &+ &- | left_associative | |
| >> << | left_associative | |
| < > <= >= | left_associative | |
| == != | left_associative | |
| & | left_associative | |
| ^ | left_associative | |
| \| | left_associative | |
| && | left_associative | |
| \|\| | left_associative | |
| ? : | right_associative | LOWEST |

The "border" parameter in the eval() command determines the border handling when doing indexed image access. For instance the command:

```
      eval "B=A[-10,0]", 0
```

tries to access pixels with x coordinate less than zero for xx<10. If "border" equals -1 the image is mirrored in its borders. This option is useful for linear filters. The second option is to set border to a value (any value but -1). Now pixels outside the actual image are assumed to have the "border" value. This option is most often used in non-linear filters.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

## *even_ok*

***NAME***

even_ok - check if a value is a even integer value

***SYNOPSIS***

```
#include "im_infra.h"

int even_ok(int value, char *text)
```

***DESCRIPTION***

The parameter "value" is checked to see if it is an even value. If it is not an even value an error is generated and the following message is added to the error-stack:

```
      <text> [<value>] should be even
```

***RETURN VALUES***

IE_OK (1) if the value is even
IE_NOT_OK (0) if the value is odd

***SEE ALSO***

odd_ok

## *EventType*

***NAME***

EventType - obtain the type of an event

***SYNOPSIS***

```
#include "disp_p.h"

int EventType(IM_EVENT event)
```

***DESCRIPTION***

EventType can be used to find out the type of an obtained "event". The type returns the type of event as one of the symbolic values:

PRESS_EVENT
RELEASE_EVENT
MOVE_EVENT
KEYBOARD_EVENT

These symbolic values are defined in the include file "imwindow.h"

EventType can only be used after a call to the "point_im" routine which returns an event as one of its arguments.

***EXAMPLE***

```
#include "disp_p.h"
#include "imwindow.h"
#include "image.h"

IMAGE    *ip;
int       x, y;
int       val;
IM_EVENT  event;

while (point_im(&ip, &x, &y, &event) != ’q’) {
      val = EventType(event);
      if (val == PRESS_EVENT) printf("Press\n");
      if (val == RELEASE_EVENT) printf("Release\n");
      if (val == MOVE_EVENT) printf("Move\n");
      if (val == KEYBOARD_EVENT) printf("Key\n");
}
```

***RETURN VALUES***

The type of the event as defined in the include file "imwindow.h"

PRESS_EVENT
RELEASE_EVENT
MOVE_EVENT
KEYBOARD_EVENT

***SEE ALSO***

point_im  MousePress  MouseRelease  MouseMove  IsMouseDown  KeyPressed

## *exit*

## *_exit*

***NAME***

exit, _exit - terminate process

***SYNOPSIS***

```
void exit(int status)

void _exit(int status)
```

***DESCRIPTION***

These functions are interface functions to the standard C functions as implemented on the current operating system. The functionality of these functions is:

exit() is the normal means of terminating a process. Exit closes all the process’s files and notifies the parent process if it is executing a wait. The low-order 8 bits of status are available to the parent process.

This call can never return.

The C function exit may cause cleanup actions before the final "sys exit". The function "_exit" circumvents all cleanup, and should be used to terminate a child process after a fork(2) to avoid flushing buffered output twice.

***RETURN VALUES***

The function does not return

## *exp*

## *log*

## *log10*

## *pow*

## *sqrt*

***NAME***

exp, log, log10, pow, sqrt - exponential, logarithm, power, square root

***SYNOPSIS***

```
#include <math.h>

double exp(double x)

double log(double x)

double log10(double x)

double pow(double x, double y)

double sqrt(double x)
```

***DESCRIPTION***

These functions are interface functions to the standard C functions as implemented on the current operating system. The functionality of these functions is:

exp returns the natural exponent function of "x".

log returns the natural logarithm of "x".

log10 returns the base 10 logarithm.

pow returns "x"**"y", "x" to the "y" power.

sqrt returns the square root of "x".

***RETURN VALUES***

exp() and pow() return a huge value when the correct value would overflow; errno is set to ERANGE.

pow() returns 0 and sets errno to EDOM when the second argument is negative and nonintegral and when both arguments are 0.

log() returns 0 when "x" is zero or negative; errno is set to EDOM.

sqrt() returns 0 when "x" is negative; errno is set to EDOM.

***SEE ALSO***

hypot  sinh

---

## *exp_im*

***NAME***

exp_im - natural exponentiation

***SYNOPSIS***

```
#include "im_proto.h"

int exp_im(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

Raise the base "e" to the power of each element of "in" and store the result in the corresponding element of "out".

***NOTE***

For more powerful image arithmetic expressions (scaling, adding offsets, etc.), use the function eval().

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

exp10_im  ln_im

---

## *exp10_im*

***NAME***

exp10_im - 10 based exponentiation

***SYNOPSIS***

```
#include "im_proto.h"

int exp10_im(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

Raise the base 10 to the power of each element of "in" and store the result in the corresponding element of "out".

***NOTE***

For more powerful image arithmetic expressions (scaling, adding offsets, etc.), use the function eval() .

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

exp_im  ln_im  log10_im

## *expand*

***NAME***

expand - enable/disable command expanding

***SYNOPSIS***

```
expand <mode>
```

***DESCRIPTION***

The SCIL environment is provided with a command expander, which forms a command layer on top of the C interpreter syntax.

In the command expander mode commands may be abbreviated, and need no C punctuation. If arguments are missing defaults resident in a commando description file will be inserted. All given arguments are checked for legitimate values.

With a question mark "?" as an argument a prompt range and default is generated.

"mode" is 1 enable the command expansion, "mode" is 0 disable it.

***EXAMPLE***

```
[C1] expand 1
[C2] readf ?
Filename [ trui.im ] :
Image <A|B|C|D> [ A ] :
[C3] expand 0
[C4] readf ?
readf ?--> variable used but not declared
[C5]
```

## *fabs*

## *floor*

## *ceil*

***NAME***

fabs, floor, ceil - absolute value, floor, ceiling functions

***SYNOPSIS***

```
#include <math.h>

double floor(double x)

double ceil(double x)

double fabs(double x)
```

***DESCRIPTION***

These functions are interface functions to the standard C functions as implemented on the current operating system. The functionality of these functions is:

fabs() returns the absolute value |"x"|.

floor() returns the largest integer not greater than "x".

ceil() returns the smallest integer not less than "x".

***SEE ALSO***

abs

## *fast_fourier*

***NAME***

fast_fourier - Fast Fourier Transform

***SYNOPSIS***

```
#include "im_proto.h"

int fast_fourier(IMAGE *in, IMAGE *out, int direction)
```

***DESCRIPTION***

Calculate the forward or reverse fourier transform of the image "in" and store the result in the image "out". The forward transform is performed if direction = 1, and the reverse is performed if direction = 0. The algorithm is based upon the radix-2 fast Fourier transform. The result is scaled with a factor 1/sqrt(M*N), sqrt being the square root and M and N being the dimension of the operand in x- and y-direction. The definition of the frequency domain representation corresponds with the so-called "optical Fourier transform", which means that the frequency components range from -fs/2 to +fs/2 and the zero frequency resides in the center of the image "out", fs being the spatial sampling frequency in the corresponding direction. If the pixel coordinates within the image where the frequency domain representation resides range from

i = 0 to N-1 for the x-direction and
j = 0 to M-1 for the y-direction,

the pixel with coordinates (i,j) corresponds with the frequency components:

fx = (i - N/2) / (N * dx) and
fy = (j - N/2) / (N * dy),

dx and dy being the sample intervals in the x-and y-direction in the image where spatial domain representation resides.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

fast_hartley

## *fast_hartley*

***NAME***

fast_hartley - Fast Hartley Transform

***SYNOPSIS***

```
#include "im_proto.h"

int fast_hartley(IMAGE *in, IMAGE *out, int norescaling)
```

***DESCRIPTION***

Calculate the Hartley transform of the image "in" and store the result in the image "out". The Hartley transform is symmetric; the forward transform is the same as its reverse. The algorithm is based upon the radix-2 fast Hartley transform. The result is scaled with a factor 1/sqrt(M*N),M and N being the dimension of the operand in x- and y-direction. If "norescaling" is True (1), no scaling is performed.

The definition of the frequency domain representation corresponds with the so-called "optical Fourier transform", which means that the frequency components range from -fs/2 to +fs/2 and the zero frequency resides in the center of the image "out", fs being the spatial sampling frequency in the corresponding direction. If the pixel coordinates within the image where the frequency domain representation resides range from

i = 0 to N-1 for the x-direction and j = 0 to M-1 for the y-direction,

the pixel with coordinates (i,j) corresponds with the frequency components:

fx = (i - N/2) / (N * dx) and fy = (j - N/2) / (N * dy),

dx and dy being the sample intervals in the x-and y-direction in the image where spatial domain representation resides.

***LITERATURE***

R.N. Bracewell, "Discrete Hartley Transform", Optical Society of America, pp.1832-1835, 1983.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

fast_fourier

## *fblow*

***NAME***

fblow - interpolating image blow-up

***SYNOPSIS***

```
#include "im_proto.h"

int fblow(IMAGE *in, IMAGE *out, double hfact, double vfact, double
dfact, int adjust)
```

***DESCRIPTION***

Blow image "in" with a horizontal factor "hfact", a vertical factor "vfact" and depth factor "dfact" by linear interpolation and store the result in image "out". The sizes of the output image "out" are not adjusted to fit the result, only when the parameter "adjust" is true (not zero), the sizes of "out" will be adjusted.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

blow  reduce

## *fclose*

## *fflush*

***NAME***

fclose, fflush - close or flush a stream

***SYNOPSIS***

```
#include <stdio.h>

int fclose(FILE *stream)

int fflush(FILE *stream)
```

***DESCRIPTION***

These functions are interface functions to the standard C functions as implemented on the current operating system. The functionality of these functions is:

fclose() causes any buffers for the named "stream" to be emptied, and the file to be closed. Buffers allocated by the standard input/output system are freed to be used with another fopen.

fclose() is performed automatically upon calling exit(2).

fflush() causes any buffered data for the named output "stream" to be written to that file. The stream remains open.

***RETURN VALUES***

These routines return EOF if stream is not associated with an output file, or if buffered data cannot be transferred to that file.

***SEE ALSO***

close  fopen  setbuf

---

## *fft*

***NAME***

fft

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See fast_fourier

---

## ***fg_buffers***

## ***fgr_buffers***

***NAME***

fg_buffers, fgr_buffers - get the number of available image buffers

***PLATFORM***

MS-Windows.

***SYNOPSIS***

```
#include "scilgrab.h"

int fg_buffers(void)

int WINAPI fgr_buffers(void)
```

***DESCRIPTION***

fg_buffers() returns the number of buffers of the grabber that can be used to grab images to. Framegrabbers may be able to rearrange their available memory depending on the size of the grab-region and the roi settings. So when changing the sizes of the region with fg_setres() or fg_setroi(), the number of available image buffers may change.

fgr_buffers(): see fg_buffers().

***RETURN VALUES***

the number of available image buffers.
0 on failure

***SEE ALSO***

fg_grab

## *fg_channels*

## *fgr_channels*

***NAME***

fg_channels, fgr_channels - get the number of available channels

***PLATFORM***

MS-Windows.

***SYNOPSIS***

```
#include "scilgrab.h"

int fg_channels(void)

int WINAPI fgr_channels(void)
```

***DESCRIPTION***

fg_channels() returns the number of available video input channels. An RGB signal is defined to be one channel of type FG_TYPE_RGB.

fgr_channels() : see fg_channels().

***RETURN VALUES***

The number of available channels on success
0 on failure

***SEE ALSO***

fg_type

## *fg_close*

## *fgr_close*

***NAME***

fg_close, fgr_close - close the frame grabber

***PLATFORM***

MS-Windows.

***SYNOPSIS***

```
#include "scilgrab.h"

int fg_close(void)

int WINAPI fgr_close(void)
```

***DESCRIPTION***

fg_close() closes the framegrabber and tries to unload the grabber DLL from memory

fgr_close() must close/reset the grabber. Most grabbers require some cleanup, freeing of memory and releasing of the hardware to reset the grabber board to a state it can be used by other application.

***RETURN VALUES***

1 on success
0 on failure

***SEE ALSO***

fg_load  fg_init

## *fg_depth*

## *fgr_depth*

## *fg_maxdepth*

## *fgr_maxdepth*

***NAME***

fg_depth, fgr_depth - get the pixel depth of the grabber

fg_maxdepth, fgr_maxdepth - get the maximum pixel depth of the grabber

***PLATFORM***

MS-Windows.

***SYNOPSIS***

```
#include "scilgrab.h"

int fg_depth(void)

int WINAPI fgr_depth(void)

int fg_maxdepth(void)

int WINAPI fgr_maxdepth(void)
```

***DESCRIPTION***

fg_depth() returns the actual pixel depth from the grabber.

fg_maxdepth() returns the maximum pixel depth the grabber can grab at. For full color grabbers (RGB grabbers), the pixel depth for one color component is returned.

fgr_depth(): see fg_depth().

fgr_maxdepth(): see fg_maxdepth().

***RETURN VALUES***

fg_depth (fgr_depth) returns the actual pixel depth of the grabber
fg_maxdepth (fgr_maxdepth) returns the maximum pixel depth of the grabber.

***SEE ALSO***

fg_setdepth

## *fg_exec*

## *fgr_exec*

***NAME***

fg_exec, fgr_exec - execute grabber specific function

***PLATFORM***

MS-Windows.

***SYNOPSIS***

```
#include "scilgrab.h"

int fg_exec(const void *input, int ilength, void *output, int
olength)

int WINAPI fgr_exec(const void *input, int ilength, void *output, int
olength)
```

***DESCRIPTION***

fg_exec() is a hook to call grabber specific functions. The contents and layout of the parameters are undefined and left to the programmer of the DLL. Usage of this function is inherently not portable between grabbers.

If any data is transported to or from the grabber to host memory, this must be indicated by the "input" and "output" pointers and "ilength" and "olength" counts. The return value must then be the amount of bytes transported to "output" (in a 16 bits DLL the data is buffered and "input" and "output" together may not exceed 64 KB).

fgr_exec(): see fg_exec().

***RETURN VALUES***

any value except 0 is considered a good-status.
0 on failure

## *fg_gain*

## *fgr_gain*

## *fg_setgain*

## *fgr_setgain*

***NAME***

fg_gain, fgr_gain - retrieve the gain of the video signal

fg_setgain, fgr_setgain - set the gain of the video signal

***PLATFORM***

MS-Windows.

***SYNOPSIS***

```
#include "scilgrab.h"

int fg_gain(int *pgain)

int WINAPI fgr_gain(int *pgain)

int fg_setgain(int gain)

int WINAPI fgr_setgain(int gain)
```

***DESCRIPTION***

fg_setgain() sets the gain of the video signal before A/D conversion. The actual gain value to use is "gain"/256. So "gain" =256 means a gain of 1.0 for the grabber. Input values will be converted to appropriate values for the grabber. Input values out of range will be clipped to the maximum or minimum board setting.

fg_gain() retrieves the gain used for the video signal and stores it the integer pointed to by "pgain". The actual gain value is "*pgain"/256.

fgr_setgain() : see fg_setgain().

fgr_gain(): see fg_gain().

***RETURN VALUES***

1 on success
0 on failure

***SEE ALSO***

fg_offset  fg_setoffset

## *fg_get_datasize*

## *fgr_get_datasize*

***NAME***

fg_get_datasize, fgr_get_datasize - get the size of one pixel in bits

***PLATFORM***

MS-Windows.

***SYNOPSIS***

```
#include "scilgrab.h"

int fg_get_datasize(void)

int WINAPI fgr_get_datasize(void)
```

***DESCRIPTION***

fg_get_datasize() returns the number of bits that one pixel from the grabber occupies in memory rounded to byte size. E.g. a 12 bits grey-value grabber returns 16 (two bytes), a full color grabber with 8 bits per color and a pad-byte returns 32. If the grabber supports grabbing at different depths, the datasize of the currently set depth is returned.

fgr_get_datasize(): see fg_get_datasize()

***RETURN VALUES***

The memory requirement for one pixel of the grabber

***SEE ALSO***

fg_type  fg_depth

## *fg_get_last_error*

## *fgr_get_last_error*

***NAME***

fg_get_last_error, fgr_get_last_error - get error message on last error

***PLATFORM***

MS-Windows.

***SYNOPSIS***

```
#include "scilgrab.h"

const char *fg_get_last_error(void)

int WINAPI fgr_get_last_error(char *storage)
```

***DESCRIPTION***

fg_get_last_error() returns a pointer to a static string buffer containing a message on the last occurred error. The buffer is not overwritten until the next call to a framegrabber function.

fgr_get_last_error() must store a string describing the last occurred error in the buffer pointed to by "storage". "storage" can contain a string of maximum 256 bytes (including terminator). The DLL must store its error messages in such a way that it can at any time retrieve the string describing the last occurred error. It is recommended for the DLL to declare a static buffer "static char fgr_error_message[256]" and store the error message in that buffer. See also the function fgr_set_error_message() in the include file "scilgrab.h". fgr_get_last_error() then only has to copy the local buffer to the "storage".

***RETURN VALUES***

1 on success
0 on failure

## *fg_get_rgb_order*

## *fgr_get_rgb_order*

***NAME***

fg_get_rgb_order, fgr_get_rgb_order - get the memory layout of a RGB triplet

***PLATFORM***

MS-Windows.

***SYNOPSIS***

```
#include "scilgrab.h"

int fg_get_rgb_order(void)

int WINAPI fgr_get_rgb_order(void)
```

***DESCRIPTION***

fg_get_rgb_order() returns the memory layout a RGB triplet of a full color framegrabber. Possible values are

| | |
|---|---|
| FG_ORDER_RGB: | the RGB values are stored contiguously in memory, no padding. |
| FG_ORDER_RGBX,<br>FG_ORDER_XRGB: | the RGB triplets are stored using an extra pad-byte before or after the triplet. |
| FG_ORDER_BGRX,<br>FG_ORDER_XBGR: | the RGB triplets are stored in reverse order using an extra pad-byte before or after the triplet. |

fgr_get_rgb_order(): see fg_get_rgb_order().

***NOTE***

Currently, the SCIL_Image interactive grab-windows only support the native SCIL_Image RGB triplet layout: FG_ORDER_RGBX.

***RETURN VALUES***

The layout of the RGB triplets on success
0 on failure

***SEE ALSO***

fg_get_datasize

## *fg_getdata*

## *fgr_getdata*

***NAME***

fg_getdata, fgr_getdata - retrieve imagedata from the grabber

***PLATFORM***

MS-Windows.

***SYNOPSIS***

```
#include "scilgrab.h"

int fg_getdata(int framenumber, void *storage, int offsetx, int
offsety, int incrx, int incry, int countx, int county, int type)

int WINAPI fgr_getdata(int framenumber, void *storage, int offsetx,
int offsety, int incrx, int incry, int countx, int county, int type)
```

***DESCRIPTION***

fg_getdata() copies data from frame "framenumber" in the grabbers memory to host memory pointed to by "pstorage". Which pixels must be copied is described by the offsets ("offsetx" , "offsety"), increments ("incrx", "incry") and counts ("countx", "county"). The data will be copied row by row, first the row with the lowest offset. Positions in a frame are calculated with respect to the upper-left corner of the roi and the increments of the roi (see fg_setroi()) must be multiplied with the increments of fg_getdata().

"type" indicates the type of data to be transferred, for a list of possible value see fg_type(). The type must correspond with the type as previously set with fg_settype() or be a "subtype" of that type. E.g. if the type was set to FG_TYPE_RGB, it is also allowed to request for FG_TYPE_GREY (grey value can be calculated from RGB[1]) or just one color component like FG_TYPE_RED. For grey value grabbers, only FG_TYPE_GREY is allowed.

The depth of the transported data must be the same as set with fg_setdepth(). In the destination buffer "pstorage", the data must be aligned to byte-boundary. E.g. a 12 bits grabber must convert its data to 16 bits data (see also fg_depth(), fg_setdepth() and fg_get_datasize()).

fgr_getdata(): see fg_getdata().

[1] grey value (intensity) = 0.299 * red + 0.587 * green + 0.114 * blue.

***RETURN VALUES***

The number of retrieved scanlines on success
0 on failure

***SEE ALSO***

fg_grab_image  fg_grab  fg_type  fg_setroi

## *fg_grab*

## *fgr_grab*

## *fg_freeze*

## *fgr_freeze*

***NAME***

fg_grab, fgr_grab - start the grabber

fg_freeze, fgr_freeze - stop the grabber

***PLATFORM***

MS-Windows.

***SYNOPSIS***

```
#include "scilgrab.h"

int fg_grab(int firstchan, int firstbuf, int nchannels)

int WINAPI fgr_grab(int firstchan, int firstbuf, int nchannels)

int fg_freeze(void)

int WINAPI fgr_freeze(void)
```

***DESCRIPTION***

fg_grab() starts the grabber to grab continuously, grabbing "nchannels" input channels ( channel "firstchan", "firstchan" +1, ..) simultaneously. The result must be stored in the framebuffers "firstbuf", "firstbuf" + 1,.. Channel and buffer numbers start from zero. Grabbing continues until the function fg_freeze() is called.

fgr_grab() : see fg_grab().

fgr_freeze() : see fg_freeze().

***RETURN VALUES***

1 on success
0 on failure

***SEE ALSO***

fg_getdata  fg_grab_series  fg_grab_next

## *fg_grab_image*

***NAME***

fg_grab_image - grab an image from the framegrabber

***PLATFORM***

MS-Windows.

***SYNOPSIS***

```
#include "scilgrab.h"
#include "image.h"

int fg_grab_image(IMAGE *out, int chan, int width, int height, int
slices, int hstep, int vstep, int tstep, int offx, int offy)
```

***DESCRIPTION***

fg_grab_image() grabs an image from the framegrabber and stores it in image "out". The image is grabbed from the channel "chan". "width" and "height" indicate the X and Y size of the region to be grabbed. "slices" is the number of frames that should be grabbed. If "width", "height" or "slices" is set to 0, the corresponding size of the output image "out" is taken for that parameter. "hstep" and "vstep" can be used to skip pixels in the X and Y direction. E.g. hstep =  1 means using all the pixels from the grabber, hstep = 2 means using  every second pixels from the grabber, etc.  "tstep" can be used to skip entire frames. "offx" and "offy" indicate the offset from the upper-left corner of the grabber-image.

***RETURN VALUES***

None

***SEE ALSO***

fg_grab  fg_getdata

## *fg_grab_next*

## *fgr_grab_next*

***NAME***

fg_grab_next, fgr_grab_next - grab next frame using current settings

***PLATFORM***

MS-Windows.

***SYNOPSIS***

```
#include "scilgrab.h"

int fg_grab_next(void)

int WINAPI fgr_grab_next(void)
```

***DESCRIPTION***

Some grabbers do not support continues grabbing, they can only grab one frame and then must be given an new command to grab another frame. fg_grab_next() grabs a frame using all the current settings.

fgr_grab_next() : see fg_grab_next().

***NOTE***

Framegrabbers that do support continues grabbing should not implement this function in the DLL.

***RETURN VALUES***

1 on success
0 on failure

***SEE ALSO***

fg_getdata  fg_grab

## *fg_grab_series*

## *fgr_grab_series*

***NAME***

fg_grab_series, fgr_grab_series - grab a series of images

***PLATFORM***

MS-Windows.

***SYNOPSIS***

```
#include "scilgrab.h"

int fg_grab_series(int startchannel, int startbuffer, int nchannels,
int nbuffers, int nskip)

int WINAPI fgr_grab_series(int startchan, int startbuf, int
nchannels, int nbuffers, int nskip)
```

***DESCRIPTION***

fg_grab_series() grabs a number of images as fast as possible. The channels "startchan", "startchan"+1, ... will be read "nbuf" times and put into "startbuf", "startbuf"+1, .... "nskip" gives the possibility to slow down the process. After grabbing a frame for all channels, the function waits for "nskip" frames before grabbing again a frame for all channels. Actually the function executes "nskip" dummy grabs.

Channels and buffers are numbered 0, 1, ...

Reading can be done in one of two modes:
- FG_GRAB_CHANNELS :
  The frames are stored in the following order:
    - all frames of channel "startchan"
    - all frames of channel "startchan"+1
  - ...

- FG_GRAB_FRAMES
  The frames are stored in the order:
    - the frames of channel "startchan", "startchan"+1, …. for time 1
    - the frames of channel "startchan", "startchan"+1, …. for time 2
    - ...

fgr_grab_series(): see fg_grab_series

***RETURN VALUES***

Either FG_GRAB_CHANNELS or FG_GRAB_FRAMES on success
0 on failure

***SEE ALSO***

fg_grab  fg_getdata

## *fg_init*

## *fgr_init*

***NAME***

fg_init, fgr_init - loads the DLL and initialize the frame-grabber

***PLATFORM***

MS-Windows.

***SYNOPSIS***

```
#include "scilgrab.h"

int fg_init(const char *initfile)

int WINAPI fgr_init(const char *initfile)
```

***DESCRIPTION***

fg_init() tries to load the DLL specified with the fg_load() command and when successful initializes the frame-grabber. "initfile" is the name of a file in which grabber specific configurations are stored that may be needed by the frame-grabber. In most cases this file will typically be a camera-configuration file. The layout and contents of the file is defined by the creator of the DLL. If "initfile" == NULL (or an empty string), the grabber will be initialized with default setting.

fgr_init(): see fgr_init().

***RETURN VALUES***

Version number of the framegrabber API (SCIL_GRAB_VERSION from scilgrab.h)

***SEE ALSO***

fg_load  fg_close

## *fg_load*

***NAME***

fg_load - specify grabber DLL

***PLATFORM***

MS-Windows.

***SYNOPSIS***

```
#include "scilgrab.h"

int fg_load(const char *dllname)
```

***DESCRIPTION***

Specify the name of the framegrabber DLL that is to be used in all subsequent fg_…. functions.

***NOTE***

The framegrabber DLL is not actually loaded until the next call to a framegrabber function (fg_…. function)

***RETURN VALUES***

Always 1

***SEE ALSO***

fg_close

## *fg_maxwidth*

## *fgr_maxwidth*

## *fg_maxheight*

## *fgr_maxheight*

***NAME***

fg_maxwidth,  fgr_maxwidth - get the maximum allowed image width of  the grabber

fg_maxheight,  fgr_maxheight - get the maximum allowed image height of  the grabber

***PLATFORM***

MS-Windows.

***SYNOPSIS***

```
#include "scilgrab.h"

int fg_maxwidth(void)

int WINAPI fgr_maxwidth(void)

int fg_maxheight(void)

int WINAPI fgr_maxheight(void)
```

***DESCRIPTION***

fg_maxwidth() and fg_maxheight() return the maximum allowed image sizes (width and height) of the grabber. The sizes of the images that can be grabbed usually depend on the supplied video signal and/or the attached camera.

fgr_maxwidth() and fgr_maxheight(): see fg_maxwidth() and fg_maxheight()

***RETURN VALUES***

fg_maxwidth (fgr_maxwidth) returns the maximum allowed image width
fg_maxheight (fgr_maxheight) returns the maximum allowed image height

***SEE ALSO***

fg_width  fg_minwidth  fg_height  fg_minheight  fg_setres

## ***fg_minwidth***

## ***fgr_minwidth***

## ***fg_minheight***

## ***fgr_minheight***

***NAME***

fg_minwidth, fgr_minwidth - get the minimum allowed image width of the grabber

fg_minheight, fgr_minheight - get the minimum allowed image height of the grabber

***PLATFORM***

MS-Windows.

***SYNOPSIS***

```
#include "scilgrab.h"

int fg_minwidth(void)

int WINAPI fgr_minwidth(void)

int fg_minheight(void)

int WINAPI fgr_minheight(void)
```

***DESCRIPTION***

fg_minwidth() and fg_minheight() return the minimum image sizes (width and height) of the grabber. Often grabbers have a hardware related minimum size of the images they can grab.

fgr_minwidth() and fgr_minheight(): see fg_minwidth() and fg_minheight()

***RETURN VALUES***

fg_minwidth (fgr_minwidth) returns the minimum allowed image width.
fg_minheight (fgr_minheight) returns the minimum allowed image height.

***SEE ALSO***

fg_width fg_maxwidth fg_height fg_maxheight fg_setres

## *fg_offset*

## *fgr_offset*

## *fg_setoffset*

## *fgr_setoffset*

***NAME***

fg_offset, fgr_offset - retrieve the offset of the video signal

fg_setoffset, fgr_setoffset - set the offset of the video signal

***PLATFORM***

MS-Windows.

***SYNOPSIS***

```
#include "scilgrab.h"

int fg_offset(int *poffset)

int WINAPI fgr_offset(int *poffset)

int fg_setoffset(int offset)

int WINAPI fgr_offset(int offset)
```

***DESCRIPTION***

fg_setoffset() sets the offset of the video signal before A/D conversion. The offset value is specified in millivolts (mV).  Input values out of range will be clipped to the maximum or minimum board setting.

fg_offset() retrieves the offset used for the video signal and stores it the integer pointed to by "poffset". The offset value is given in millivolts (mV).

fgr_setoffset() : see fg_setoffset().

fgr_offset(): see fg_offset().

***RETURN VALUES***

1 on success
0 on failure

***SEE ALSO***

fg_gain  fg_setgain

## *fg_set_input_lut*

## *fgr_set_input_lut*

***NAME***

fg_set_input_lut, fgr_set_input_lut - set input lookup table

***PLATFORM***

MS-Windows.

***SYNOPSIS***

```
#include "scilgrab.h"

int fg_set_input_lut(const void *ptable, int nelem)

int WINAPI fgr_set_input_lut(const void *ptable, int nelem)
```

***DESCRIPTION***

fg_set_input_lut() loads an input lookup table into the grabbers memory. "ptable" points to an array of "nelem" pixel values. For 8-bits grabbers the input values must be of type "unsigned char". For a grabber up to 16 bits (but over 8-bits), the elements must be of type "unsigned short" etc. If  "nelem" is less the actual length of the input lookup-table, remaining values must be set to zero

fgr_set_input_lut(): see fg_set_input_lut().

***RETURN VALUES***

1 on success
0 on failure

## *fg_setdepth*

## *fgr_setdepth*

***NAME***

fg_setdepth, fgr_setdepth - set the pixeldepth to a given value

***PLATFORM***

MS-Windows.

***SYNOPSIS***

```
#include "scilgrab.h"

int fg_setdepth(int depth)

int WINAPI fgr_setdepth(int depth)
```

***DESCRIPTION***

fg_setdepth() sets the pixeldepth of the grabber to "depth", if the grabber cannot grab at the given depth, another depth (determined by the grabber) is used, which is also returned. For full color grabbers, the depth of one color component is returned.

fgr_setdepth(): see fg_setdepth(). If the grabber hardware does not support the requested depth, the depth must be set to a value the grabber can support and this value must be returned.

***RETURN VALUES***

The new pixeldepth (may be different than the one supplied)

***SEE ALSO***

fg_depth  fg_maxdepth

## *fg_setres*

## *fgr_setres*

***NAME***

fg_setres, fgr_setres - set the sizes of the image in the grabber

***PLATFORM***

MS-Windows.

***SYNOPSIS***

```
#include "scilgrab.h"

int fg_setres(int width, int height)

int WINAPI fgr_setres(int width, int height)
```

***DESCRIPTION***

fg_setres() sets the sizes of the image to grab to "width" and "height". These sizes may be rounded  by the grabber to a higher value (not exceeding the maximum allowed sizes) if the grabber hardware does not support the given sizes. This can be checked with fg_width() and fg_height(), these should then return the sizes the grabber can support.

fgr_setres(): see fg_setres(). If the grabber hardware does not support the given size, the sizes must be converted to the nearest higher size the grabber can support, off course never exceeding the maximum allowed sizes of the grabber.

***RETURN VALUES***

1 on success
0 on failure

***SEE ALSO***

fg_setroi  fg_width  fg_height  fg_minwidth  fg_maxwidth  fg_minheight fg_maxheight

## *fg_setroi*

## *fgr_setroi*

***NAME***

fg_setroi, fgr_setroi - set region of interest in the image

***PLATFORM***

MS-Windows.

***SYNOPSIS***

```
#include "scilgrab.h"

int fg_setroi(int ox, int oy, int ix, int iy, int cx, int cy)

int WINAPI fgr_setroi(int ox, int oy, int ix, int iy, int cx, int cy)
```

***DESCRIPTION***

fg_setroi() specifies within the sizes of the grabber image a roi (Region Of Interest) to be used for data acquisition. The roi start at position ("ox", "oy") from the upper-left corner of the image. "cx" and "cy" are the number of pixels in the X and Y direction in the ROI and "ix" and "iy" are the increments i.e. the distance between two pixels of the ROI in the image itself. E.g. "ix" = 2 means that every second pixel in the X direction of  the image is part of the ROI.

fgr_setroi(): see fg_setroi().

***RETURN VALUES***

1 on success
0 on failure

***SEE ALSO***

fg_getdata  fg_setres

## *fg_type*

## *fgr_type*

## *fg_settype*

## *fgr_settype*

***NAME***

fg_type, fgr_type - get the image type of the grabber

fg_settype, fgr_settype - set the image type of the grabber

***PLATFORM***

MS-Windows.

***SYNOPSIS***

```
#include "scilgrab.h"

int fg_type(int *ptype)

int WINAPI fgr_type(int *ptype)

int fg_settype(int type)

int WINAPI fgr_settype(int type)
```

***DESCRIPTION***

The images type supported for grabbing are:

| | |
|---|---|
| FG_TYPE_GREY: | grab grey value images, only type a grey value grabber can support. |
| FG_TYPE_RGB: | grab full color images, the default for a color grabber. |
| FG_TYPE_RED,<br>FG_TYPE_GREEN,<br>FG_TYPE_BLUE: | grab a single color component from an RGB video signal. |

fg_type() retrieves the grab type the grabber is set at, the value is returned in the integer pointed to by "ptype".

fg_settype() sets the grab type of the grabber board to "type" -which must be on of the above listed value-. If the grabber does not support the requested grab type, the function sets the grab type to a value it can support and returns that value.

fgr_type(): see fg_type()

fgr_settype(): see fg_settype()

***RETURN VALUES***

fg_settype (and fgr_settype) returns the grab type actually set or
0 on failure

fg_type (and fgr_type) returns 1 on success and
0 on failure

***SEE ALSO***

fg_getdata

## *fg_width*

## *fgr_width*

## *fg_height*

## *fgr_height*

***NAME***

fg_width,  fgr_width - get the actual image width of the grabber

fg_height,  fgr_height - get the actual image height of the grabber

***PLATFORM***

MS-Windows.

***SYNOPSIS***

```
#include "scilgrab.h"

int fg_width(void)

int WINAPI fgr_width(void)

int fg_height(void)

int WINAPI fgr_height(void)
```

***DESCRIPTION***

fg_width() and fg_height() return the actual settings of the width and height of the image in the grabber.

fgr_width() and fgr_height(): see fg_width() and fg_height().

***RETURN VALUES***

fg_width (fgr_width) returns the image width
fg_height (fgr_height) returns the image height

***SEE ALSO***

fg_setres  fg_minwidth  fg_maxwidth  fg_minheight  fg_maxheight

## *fgetpos*

## *fsetpos*

***NAME***

fgetpos, fsetpos - manipulate pointer position in a stream

***SYNOPSIS***

```
#include <stdio.h>

int fgetpos(FILE *stream, fpos_t *ptr)

int fsetpos(FILE *stream, fpos_t *ptr)
```

***DESCRIPTION***

This function is an interface to the standard C function as implemented on the current system. The functionality of this function is:

fgetpos() records the current position in "stream" in "*ptr", for subsequent use by fsetpos() . The type fpos_t is suitable for recording such values.

fsetpos() positions "stream" at the position recorder by fgetpos() in "*ptr".

***RETURN VALUES***

fgetpos() and fsetpos() return non-zero on error.

***SEE ALSO***

ftell  fseek

## *fgreater0_ok*

***NAME***

fgreater0_ok - check if a float value is bigger than zero

***SYNOPSIS***

```
#include "im_infra.h"

int fgreater0_ok(double value, char *text)
```

***DESCRIPTION***

The floating point value "value" is checked to see if it is greater than zero or not. If it is zero or negative  an error is generated and the following message is added to the error-stack:

```
      <text> [<value>] must be bigger than 0
```

***NOTE***

This function can only handle float values, to check on integer values, use the function greater0_ok().

***RETURN VALUES***

IE_OK (1) if the value is bigger than zero
IE_NOT_OK (0) if it is zero or negative

***SEE ALSO***

greater0_ok  fpositive_ok

## *filter_energy_ratio*

***NAME***

filter_energy_ratio - calculates discriminative power of a filter

***SYNOPSIS***

```
#include "im_proto.h"

double filter_energy_ratio(IMAGE *in1, IMAGE *in2, IMAGE *filter)
```

***DESCRIPTION***

These functions calculate the energy ratio between the images "in1" and "in2" for filter image "filter". This ratio is a measure of the discriminative power of "filter" between pattern "in1" and pattern "in2". It is assumed that the mean value of the filter is zero. The ratio is calculated by taking the ratio of the variance between both images after convolution by "filter". The ratio is corrected for the (maybe different) size of the input images.

***RETURN VALUES***

The energy ratio

***SEE ALSO***

benke

---

## *flip*

***NAME***

flip - rotate an image on X, Y or Z axis over 90, 180 or 270 degrees

***SYNOPSIS***

```
#include "im_proto.h"

int flip(IMAGE *in, IMAGE *out, int axis, int angle)
```

***DESCRIPTION***

flip() rotates the image "in" over angle of 90, 180 or 270 degrees on either the X, Y or Z axis of the image. The result is stored in the image "out". Two dimensional images can only be rotated on the Z axis. "axis" specifies the axis on which to rotate, X=1, Y=2 and Z=3. The angle "angle" must be specified in degrees, so valid angles are 90, 180 and 270.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

rotate

---

## *fmod*

## *frexp*

## *ldexp*

## *modf*

***NAME***

fmod, frexp, ldexp, modf - manipulate part of floating point numbers

***SYNOPSIS***

```
#include <math.h>

double fmod(double x, double y)

double frexp(double x, int *exp)

double ldexp(double x, int n)

double modf(double x, double *ip)
```

***DESCRIPTION***

This function is an interface to the standard C function as implemented on the current system. The functionality of this function is:

fmod() returns the floating-point remainder of "x/y", with the same sign as "x". If "y" is zero, the result is implementation dependent.

frexp() splits "x" into normalized fraction in the interval [1/2,1), which is returned, and a power of 2, which is stored in "*exp". If "x" is zero, both parts of the result are zero

ldexp() returns "x * 2 ** n"

modf() splits "x" into integral and fractional parts, each with the same sign as "x". It stores the integral part in "*ip", and returns the fractional part.

***RETURN VALUES***

see description of the functions

## *fopen*

## *freopen*

## *fdopen*

***NAME***

fopen, freopen, fdopen - open a stream

***SYNOPSIS***

```
#include <stdio.h>

FILE *fopen(char *filename, char *type)

FILE *freopen(char *filename, char *type, FILE *stream)

FILE *fdopen(int fildes, char *type)
```

***DESCRIPTION***

These functions are interface functions to the standard C functions as implemented on the current operating system. The functionality of these functions is:

fopen() opens the file named by "filename" and associates a stream with it. fopen() returns a pointer to be used to identify the stream in subsequent operations.

"type" is a character string having one of the following values:

| | |
|---|---|
| "r" | open for reading |
| "w" | create for writing |
| "a" | append: open for writing at end of file, or create for writing |

In addition, each type may be followed by a "+" to have the file opened for reading and writing.

| | |
|---|---|
| "r+" | positions the stream at the beginning of the file, |
| "w+" | creates or truncates it, |
| "a+" | positions it at the end. |

Both reads and writes may be used on read/write streams, with the limitation that an fseek(), rewind(), or reading an end-of-file must be used between a read and a write or vice-versa.

freopen() substitutes the named file in place of the open "stream". It returns the original value of "stream". The original stream is closed.

freopen() is typically used to attach the preopened constant names, stdin, stdout, stderr, to specified files.

fdopen() associates a stream with a file descriptor "fildes" obtained from open(), dup(), creat(), or pipe(2). The "type" of the stream must agree with the mode of the open file.

***BUGS***

fdopen() is not portable to systems other than UNIX.

The read/write types do not exist on all systems. Those systems without read/write modes will probably treat the type as if the "+" was not present.

***RETURN VALUES***

fopen() and freopen() return the pointer NULL if filename cannot be accessed.

***SEE ALSO***

open  fclose

---

## *fpositive_ok*

***NAME***

fpositive_ok - check if a float value is positive

***SYNOPSIS***

```
#include "im_infra.h"

int fpositive_ok(double value, char *text)
```

***DESCRIPTION***

The float value "value" is checked to see if it is not negative or not. If the value is negative  an error is generated and the following message is added to the error-stack:

```
<text> [<value>] must be positive
```

Zero is considered to be positive as well in this function, if a check must be performed on a value that may not be zero then the function fgreater0_ok() can be used.

***NOTE***

This function can only handle floating point values, to check on integer values, use the function fpositive_ok().

***RETURN VALUES***

IE_OK (1) if the value is positive (zero included)
IE_NOT_OK (0) if the value is negative

***SEE ALSO***

fgreater0_ok  positive_ok

---

## *fraction_im*

***NAME***

fraction_im - take the fractional part of pixel values

***SYNOPSIS***

```
#include "im_proto.h"

int fraction_im(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

Take from each element of the image "in" the fractional part and store the results into the corresponding element of "out". The fractional part has the same sign as the original value.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

truncate_im  nearest_int  lowest_int

## *frange_ok*

***NAME***

frange_ok - check is a float value is in the specified range

***SYNOPSIS***

```
#include "im_infra.h"

int frange_ok(double value, double vmin, double vmax, char *text)
```

***DESCRIPTION***

frange_ok() checks to see if floating point "value" is in the range specified by "vmin" and "vmax" (borders included). If it is, a true status is returned. If "value" is outside the range  an error is generated and the following message is added to the error-stack:

```
<text> [<value>] out of range (<vmin>..<vmax>)
```

A lot of the checking routines use this function to do the actual checking and supply a default message for that specific check.

***NOTE***

The function has exactly the same behavior as range_ok(), except that this function handles only floating point values and range_ok() can only handle integer values.

***RETURN VALUES***

IE_OK (1) if the value is inside the range (borders included).
IE_NOT_OK (0) if the value is outside the range.

***SEE ALSO***

range_ok  fpositive_ok  fgreater0_ok  funequal0_ok

## *fread*

## *fwrite*

***NAME***

fread, fwrite - buffered binary input/output

***SYNOPSIS***

```
#include <stdio.h>

int fread(void *ptr, int size, int nitems, FILE *stream)

int fwrite(void *ptr, int size, int nitems, FILE *stream)
```

***DESCRIPTION***

These functions are interface functions to the standard C functions as implemented on the current operating system. The functionality of these functions is:

fread reads, into a block beginning at "ptr", "nitems" of data of the type of "*ptr" from the named input "stream". It returns the number of items actually read.

If "stream" is stdin and the standard output is line buffered, then any partial output line will be flushed before any call to read(2) to satisfy the fread.

Fwrite() writes at most "nitems" of data of the type of "*ptr" beginning at "ptr" to the named output "stream". It returns the number of items actually written.

***RETURN VALUES***

fread and fwrite return 0 upon end of file or error.

***SEE ALSO***

read  write  fopen  getc  putc  gets  puts  printf  scanf

## *fseek*

## *ftell*

## *rewind*

***NAME***

fseek, ftell, rewind - reposition a stream

***SYNOPSIS***

```
#include <stdio.h>

int fseek(FILE *stream, long offset, int ptrname)

long ftell(FILE *stream)

void rewind(FILE *stream)
```

***DESCRIPTION***

These functions are interface functions to the standard C functions as implemented on the current operating system. The functionality of these functions is:

fseek() sets the position of the next input or output operation on the "stream". The new position is at the signed distance "offset" bytes from the beginning, the current position, or the end of the file, according to whether "ptrname" has the value 0, 1, or 2.

fseek() undoes any effects of ungetc.

ftell() returns the current value of the offset relative to the beginning of the file associated with the named "stream". It is measured in bytes on UNIX; on some other systems it is a magic cookie, and the only foolproof way to obtain an offset for fseek.

rewind(stream) is equivalent to fseek(stream, 0L, 0).

***RETURN VALUES***

fseek() returns -1 for improper seeks.

***SEE ALSO***

lseek  fopen

## *funequal0_ok*

***NAME***

funequal0_ok - check to see if a float value is unequal to zero

***SYNOPSIS***

```
#include "im_infra.h"

int funequal0_ok(double value, char *text)
```

***DESCRIPTION***

If the value "value" is not equal to zero, an error is generated and the following message is added to the error-stack:

```
<text> [<value>] must be unequal to 0
```

***NOTE***

This functions can only check on floating point values, to check on integer values, use the function unequal0_ok()

***RETURN VALUES***

IE_OK (1) if "value" is unequal to zero
NOT_OK (0) if it is equal to zero

***SEE ALSO***

fpositive_ok  fgreater0_ok  unequal0_ok

## *fuzzy_derivative*

## *vfuzzy_derivative*

## *fuz_width*

***NAME***

fuzzy_derivative, vfuzzy_derivative - filter to compute the fuzzy derivatives in high precision

fuz_width - determine the width of the fuzzy filter kernel

***SYNOPSIS***

```
int fuzzy_derivative(IMAGE *in, IMAGE *out, double sigmax, double
sigmay, int derix, int deriy, double accx, double accy, int fwidthx,
int fwidthy)

int vfuzzy_derivative(IMAGE *in, IMAGE *out, double sigmax, double
sigmay, double sigmaz, int derix, int deriy, int deriz, double accx,
double accy, double accz, int fwidthx, int fwidthy, int fwidthz)

int fuz_width(double sigma, int deri, double acc, int maxlen)
```

***DESCRIPTION***

fuzzy_derivative() and vfuzzy_derivative() compute the fuzzy derivatives of image "in" and store the result in image "out". For accuracy reasons the filter is computed in floating point so the output will be a float image. vfuzzy_derivative() is the 3D version of fuzzy_derivative(), the only difference is the parameter list, which is extended with parameters for the Z-dimension ("sigmaz", "accz", "deriz" and "fwidthz").

fuz_width() returns the width of the filter kernel given a  specific "sigma", order of derivative "deri" and accuracy "acc".

The filter coefficients for the different order of derivatives  are given by:

| degree of derivative | filter function |
|---|---|
| 0 | gauss(x,s) = 1/((sqrt(2*pi)*s)exp(-x**2/2*s**2), |
| 1 | (-x/s)*gauss (x,s) |
| 2 | (x**2/s**2 - 1)*gauss (x,s) |
| 3 | (-x**3/s**4 + 3*x/s**2)*gauss (x,s) |
|  | where s is the sigma. |

The filter has parameter "sigmax", "sigmay" and "sigmaz" governing the effective width of the filter, where the minimum value of 1.0  corresponds to the size of one pixel. The maximum value of sigma  is 10.0. The parameter "derix", "deriy" and "deriz" give the order of the  derivatives. They may have different values for each dimension.  For higher values of "sigma", the image can safely be analyzed at a reduced resolution.

The actual width of the filter as the length of the set of filter coefficients is governed by the parameter "accx", "accy" and "accz". It indicates how close the filter set resembles the Gauss function. In the implementation of the Gauss filter, the left and right tails are chopped off to keep a fraction of "acc" of the total filter mass. The mass fraction of 1-acc being chopped off is redistributed over the remaining coefficients to ensure proper values.

The filter outcome is normalized for constant images for no derivation (order 0), linear ramps for derivatives of order 1, parabolas for order 2, and cubes for order 3. The image is mirrored near the edges during the computation of the filter to prevent strong edge effects.

Note carefully that parameter "acc" determines the actual width of the filter. For the same value of "acc", the parameter will cause different widths for different values of "deri". This may cause undesired effects in case a mixture of derivatives is used in one formula. To that end, the parameters "fwidthx", "fwidthy" or "fwidthz" when given a positive and odd value overrule the parameter "accx", "accy" or "accz". Then, the actual filter width is the value of this parameter, regardless its accuracy. "fwidthx", "fwidhty" or "fwidthz" may not be larger then the width, the height or the depth of the image. When "fwidthx", "fwidthy" or "fwidthz" has the value of -1, the parameter "acc" will function as described above.

fuz_width() returns width (or height) of the filter kernel that is used by fuzzy_derivative() and gauss(). "sigma", "deri" and "acc" have the same meaning as the corresponding arguments of fuzzy_derivative().

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

gauss vgauss

## *g_copy_object*

***NAME***

g_copy_object - copy object grey source with object mask as reference

***SYNOPSIS***

```
#include "im_aio.h"

int g_copy_object(IMAGE *grey_src, IMAGE *mask_src, IMAGE *dst, LIST
*link)
```

***DESCRIPTION***

| | | |
|---|---|---|
| grey_src | - | Image with object grey sources |
| mask_src | - | Image with labeled objects |
| dst | - | Destination image |
| link | - | Link pointing to object |

g_copy_object() copies the object grey value sources to the same coordinate in the destination image, with help of a labeled mask image.

***EXAMPLE***

```
To copy the grey value objects not touching the edge to another
image:

#include "image.h "
#include "im_aio.h"
LIST *l, *o;

readfile("cermet",a,0,0);
threshold(a,b,128);
invert_im(b,b);
l = list_label(b,c,8,0);
FORALL(o,l)
    if (!edge_object(c, o))
        g_copy_object(a,c,d,o);
display_image(d);
l = rm_list(l);
```

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

copy_object  g_copy_object_to

## *g_copy_object_to*

***NAME***

g_copy_object_to - copy object grey source with object mask as reference to a specified coordinate

***SYNOPSIS***

```
#include "im_aio.h"

int g_copy_object_to(IMAGE *grey_src, IMAGE *mask_src, IMAGE *dst,
LIST *link, int x, int y)
```

***DESCRIPTION***

| | | |
|---|---|---|
| grey_src | - | Image with object grey sources |
| mask_src | - | Image with labeled objects |
| dst | - | Destination image |
| link | - | Link pointing to object |
| x, y | - | Coordinate X, Y |

g_copy_object_to() copies the object grey value sources to the specified coordinate in the destination image, with help of a labeled mask image.

***EXAMPLE***

```
To copy the grey value objects not touching the edge to another image
at coordinate 0,0:

#include "image.h "
#include "im_aio.h"
LIST *l, *o;

readfile("cermet",a,0,0);
threshold(a,b,128);
invert_im(b,b);
l = list_label(b,c,8,0);
FORALL(o,l)
    if (!edge_object(c, o))
        g_copy_object_to(a,c,d,o,0,0);
display_image(d);
l = rm_list(l);
```

***RETURN VALUES***

IE_OK (1) on success

Negative error status on failure (see im_error.h)

***SEE ALSO***

copy_object  g_copy_object

## *gauss*

## *vgauss*

***NAME***

gauss, vgauss - gauss filter

***SYNOPSIS***

```
#include "im_proto.h"

int gauss(IMAGE *in, IMAGE *out, double sigmax, double sigmay, double
accx, double accy, int fwidthx, int fwidthy)

int vgauss(IMAGE *in, IMAGE *out, double sigmax, double sigmay,
double sigmaz, double accx, double accy, double accz, int fwidthx,
int fwidthy, int fwidthz)
```

***DESCRIPTION***

gauss() and vgauss() perform Gauss filtering on image "in", resulting in image "out". The filter coefficient values are given by:

```
      gauss(x,s) = 1/((sqrt(2*pi)*s)exp(-x**2/2*s**2)
```

where s is the sigma

The filter has parameter "sigmax", "sigmay" and "sigmaz" governing the effective width of the filter, where the minimum value of 1.0 corresponds to the size of one pixel. The maximum value of sigma is 10.0. For higher values of sigma, the image can safely be analyzed at a reduced resolution.

The actual width of the filter as the length of the set of filter coefficients is governed by the parameter "accx", "accy" and "accz". It indicates how close the filter set resembles the Gauss function. In the implementation of the Gauss filter, the left and right tails are chopped off to keep a fraction of "acc" of the total filter mass. The mass fraction of 1-acc being chopped off is redistributed over the remaining coefficients to ensure proper values.

The filter outcome is normalized for constant images. The image is mirrored near the edges during the computation of the filter to prevent strong edge effects.

Note carefully that parameter "acc" determines the actual width of the filter. The parameters "fwidthx", "fwidthy" or "fwidthz" when given a positive and odd value overrule the parameter "accx", "accy" or "accz". Then, the actual filter width is the value of this parameter, regardless its accuracy. "fwidthx", "fwidthy" and "fwidhtz" may not be larger than the width, the height and the depth of the image. When "fwidthx", "fwidthy", "fwidthz" have the value of -1, the parameter "acc" will function as described above.

vgauss() is the 3D version of gauss(), the only difference is the parameter list, which is extended with parameters for the Z-dimension ("sigmaz", "accz" and "fwidthz").

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***
fuzzy_derivative vfuzzy_derivative

---

## *gauss_deblur*

***NAME***
gauss_deblur - gaussian deblurring enhancement

***SYNOPSIS***
```
#include "im_proto.h"

int gauss_deblur(IMAGE *in, IMAGE *out, double sigma, int order,
double accuracy, double factor)
```

***DESCRIPTION***
The image "in" is deblurred by extrapolating the taylor expansion in the scale direction. Differentiation up to order "order" is performed and a weighted sum (given by the taylor_polynomial()) of all even derivatives is stored in the image "out". The fuzzy_derivative() function is used for differentiation of the input image, with scale parameter "sigma". The extrapolation parameter "factor" determines the enhancement factor; 0.5 agrees to the natural scale. The scale parameter used for extrapolation is -"factor" * "sigma" ** 2; this results in extrapolation of scale linear with "factor". Since blurring is a destructive (semi-group) operation, this is only a enhancement technique; no new detail is created.

***LITERATURE***
L. Florack, The syntactical structure of scalar images, PhD Thesis, University of Utrecht, The Netherlands, 1991.

***RETURN VALUES***
IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***
taylor_polynomial

---

## *gauss_family*

***NAME***

gauss_family - complete set of derived gaussians

***SYNOPSIS***

```
#include "im_proto.h"

int gauss_family(IMAGE *in, IMAGE *out, double sigma, int order,
double accuracy, int zero, int even)
```

***DESCRIPTION***

Differentiate the input image" in" and store the result in each plane of the 3D-output image "out". In the output image, first the highest y derivative is put, then incremently the x derivatives. For each order, there are order+1 components. So the output result will be such as:

{f}, {f/dy , f/dx}, {f/dydy, f/dydx, f/dxdx},

Differentiation is performed by means of derived gaussian filtering, for which the scale is given by "sigma". If "zero" is true, the zero-order (smoothed) is included in the output. If "even" is true, only even orders are included. This result in:

{f}, {f/dydy, f/dxdx}, {f/dydydydy, f/dydydxdx, f/dxdxdxdx},

This expansion can be used for extrapolation of image scale.

***LITERATURE***

L. Florack, The syntactical structure of scalar images, PhD Thesis, University of Utrecht, The Netherlands, 1991.

J.J. Koenderink and A.J. van Doorn, Receptive field families, Biological Cybernetics, vol. 63, 1990, 291-297.

***RETURN VALUES***

IE_OK (1) on succes

Negative error status on failure (see im_error.h)

***SEE ALSO***

gauss_deblur  taylor_polynomial

## *geo_affine*

## *geo_rotate*

## *geo_warp*

***NAME***

geo_affine, geo_rotate, geo_warp - geometric transformations

***SYNOPSIS***

```
#include "im_proto.h"

int geo_affine(IMAGE *in, IMAGE *out, double A0, double A1, double
A2, double B1, double B2, double B3, int method, int adapt, int
border)

int geo_rotate(IMAGE *in, IMAGE *out, double angle, int method, int
adapt, int border)

int geo_warp(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

geo_affine() performs an affine geometric transformation on the image "in" and stores the result in image "out".

The affine transformation maps the pixels at coordinates (x,y) in the input image to the coordinates (x’,y’) in the output image as:

```
x’ = a0 + a1*x + a2*y
y’ = b0 + b1*x + b2*y
```

We can write this in matrix notation as:

```
(x’)   (a0)   ( a1  a2 )(x)
(y’) = (b0) + ( b1  b2 )(y)
```

As any geometric operation, the affine transformation implemented in SCIL_Image uses the backward transformation, i.e. calculating (x,y) given (x’, y’):

```
(x)   (A0)   ( A1  A2 )(x’)
(y) = (B0) + ( B1  B2 )(y’)
```

Given the forward transformation, the backward transformation reads:

```
(x)    _____1_____ (a2b0-a0b2)   _____1_____ (   b2 -a2 )(x’)
(y) =  a1b2 - a2b1 (a0b1-a1b0) + a1b2 - a2b1 ( -b1  a1 )(y’)
```

Comparing this with the definition of the  backward transform gives the values for "A0", "A1", "A2", "B0", "B1" and "B2".

If we choose a1=b2=1 a2=b1=0 we obtain:

```
(x’)   (a0)   ( 1  0 )(x)   (a0 + x)
(y’) = (b0) + ( 0  1 )(y) = (b0 + y)
```

i.e. a translation over the vector (a0,b0) from input to output image. A rotation around the origin over a degrees is given by the rotation matrix:

```
(x’)   (a0)   ( cos a  -sin a )(x)
(y’) = (b0) + ( sin a   cos a )(y)
```

Note that this is the forward transform; the backward transform equals the inverse transform. The rotation around a point (xa, ya) is obtained by first a translation such that (xa,ya) is moved to the origin, then followed by a rotation around the origin and finally the result is translated back. Thus:

```
(x’)   (x - xa)
(y’) = (y - ya)

(x’’)   ( cos a  -sin a )(x’)
(y’’) = ( sin a   cos a )(y’)

(x’’’)   (x’’ + xa)
(y’’’) = (y’’ + ya)
```

Substitution then gives an expression for (`x''',y'''`) as function of (`x,y`). This is the forward transform describing a rotation around the point (xa,ya) over a degrees. The inverse of an arbitrary affine transform has been calculated before and that gives us the backward transform.

geo_rotate() and geo_warp() are two dedicated implementations of the geo_affine() function.

geo_rotate() rotates the image "in" (or a part of it) over an arbitrary angle "angle" around the center of the image and puts the result in image "out". The rotation angle "angle" must be specified in degrees.

geo_warp() blows or reduces the image "in" to fit in the image "out". It uses bilinear interpolation for pixel calculation.

The "method" parameter specifies whether to use "bilinear interpolation" (INTERPOLATE (=1)) or "nearest neighbor" (NEAREST (=2)) pixel calculation. The size of the output image is determined by "adapt" and can be either of these values:

| | |
|---|---|
| SIZE_OF_IN (1) | output image same size as input image |
| SIZE_OF_RESULT (2) | output image large enough to contain the entire rotated image (only valid for geo_rotate()). |
| SIZE_OF_OUT (3) | output image size remains as it is, if the size is smaller than the input, not all of the image will be visible in the output. Only a part around the center of the image |

"border" determines what to do with the border in the output; either leave it black (EMPTY (=0)), or wrap the image (WRAP (=1))

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

rotate flip blow reduce warp_image fblow

---

## *get_bool_mask*

***NAME***

get_bool_mask - convert a binary image to a Boolean mask

***SYNOPSIS***

```
#include "im_infra.h"

BOOL_MASK *get_bool_mask(IMAGE *im)
```

***DESCRIPTION***

get_bool_mask() converts the specified binary image "im" into a Boolean mask. This Boolean mask can be used in the roi_define function.

***RETURN VALUES***

A pointer to the Boolean mask on success
NULL on failure

***SEE ALSO***

roi_define

---

## *get_display_mode*

***NAME***

get_display_mode - retrieve the display mode of an image

***PLATFORM***

UNIX, Macintosh.

***SYNOPSIS***

```
#include "disp_p.h"

int get_display_mode(IMAGE *image)
```

***DESCRIPTION***

get_display_mode() retrieves the display mode that is set for "image". The meaning of the values are defined in the include file "dmodes.h" Only the modes set with set_display_mode() are returned.

***RETURN VALUES***

the display mode if it is set

| | |
|---|---|
| 0 | if no display mode is set, a non existing image is specified, a roi is specified or no display window is attached to the image. |

***SEE ALSO***

set_display_mode  show_dmode_flags

---

## *get_free_entry*

***NAME***

get_free_entry - get free entry in an image-silo

***SYNOPSIS***

```
#include "silo.h"

int get_free_entry(SILOPTR siloptr)
```

***DESCRIPTION***

| | |
|---|---|
| siloptr - | Pointer to the image-silo. |

Searches the silo-entry-list for a free entry.

***RETURN VALUES***

The silo-key of the free entry is returned.
Negative error status if no free entry available (see im_error.h)

---

## *get_image_by_name*

***NAME***

get_image_by_name - obtain image pointer belonging to name

***SYNOPSIS***

```
#include "im_infra.h"

IMAGE *get_image_by_name(char *name, int case_sensitive)
```

***DESCRIPTION***

get_image_by_name() retrieves the pointer to the image whose name "name". If "case_sensitive" is non-zero upper case and lower case characters are distinct, otherwise no distinction between upper and lower case is made.

***RETURN VALUES***

NULL image "name" does not exist
a pointer to the image otherwise

***SEE ALSO***

create_image  destroy_image  roi_define

---

## *get_image_window_info*

***NAME***

get_image_window_info - get information to display in window title

***SYNOPSIS***

```
#include "im_proto.h"

int get_image_window_info(IMAGE *im, char *buf)
```

***DESCRIPTION***

get_image_window_info() puts a string in "buf"describing the type and sizes of image "im". This string is used by the display interface andplaced in the title-bar of  an image-display behind the name of the image.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

---

## *get_pixel*

## *put_pixel*

***NAME***

get_pixel, put_pixel - generic pixel value getting/putting

***SYNOPSIS***

Due to the variable number of arguments needed for different image types, the syntax is not unique.

```
#include "im_proto.h"

int get_pixel(IMAGE *im, int x, int y, [int z,] [int/double *val1,
...])

int put_pixel(IMAGE *im, int x, int y, [int z,] int/double val1,
[int/double val2, ...])
```

***DESCRIPTION***

get_pixel and put_pixel are generic functions for getting and putting pixel values in images. At the moment GREY, LABEL, BINARY, FLOAT, COMPLEX and COLOR images are supported.

The parameters between square brackets "[]" are dependent upon the image type. The first three parameters are equal for all image types, "im" a pointer to the image and the X- and Y-position "x", "y".

For all 3D images, the Z-position "z" must immediately follow the X- and Y-position (for 2D "z" may NOT be specified).

put_pixel() expects the pixel value(s) to be either an "int" or "double". "int" for the integer based image-types: BINARY, GREY, LABEL and COLOR, "double" for the floating-point based images-types FLOAT and COMPLEX. The number of values is one for BINARY, GREY, LABEL and FLOAT, two for COMPLEX and three for COLOR.

get_pixel() returns the requested pixel value dependent upon the image type using either of these two methods:

1) as its return value (an "int") for BINARY, GREY and LABEL.

2) by means of (a) pointer(s) in its parameter list FLOAT, COLOR and COMPLEX. For COLOR the number and type of  the pointers depend on the color-model, three "int" pointer for RGB, three "float" pointers for XYZ, HSI anf Lab and four "float" pointers for CMYK. One "double" pointer for FLOAT images and two "double" pointers for COMPLEX images. NULL pointers are permitted to suppress the retrieval of unwanted values.

***NOTE***

If the number and type of the parameters are not correct for the image type, the result is undefined.

***EXAMPLE***

To get the pixel value of position (95, 56) of a GREY_2D image:

```
int val;
val = get_pixel(im, 95, 56);
printf("Grey value is : %d\n", val);
```

To put the pixel value 1 to position (102, 54, 10) of a BINARY_3D image:

```
put_pixel(im, 102, 54, 10, 1);
```

To set pixel (34, 87) of a COLOR_2D image to (R,G,B) = (10,200,123):

```
put_pixel(im, 34, 87, 10, 200, 123);
```

To retrieve the value of pixel (300, 0, 5) of a COMPLEX_3D image :

```
double real, imag;
get_pixel(im, 300, 0, 5, &real, &imag);
printf("Complex value is : %g, %g\n", real, imag);
```

***RETURN VALUES***

On success either the requested value or the return status IE_OK (1) (when the value is returned through a pointer)

On failure IE_NOT_OK (0): -not implemented for image type
-position x, y [,z] is outside the image)

## *get_pixel_range*

### *NAME*

get_pixel_range - determine the range of pixel values in an image

### *SYNOPSIS*

```
#include "im_proto.h"

int get_pixel_range(IMAGE *image, double minval, double maxval)
```

### *DESCRIPTION*

get_pixel_range() calculates the range of the pixel values in the image "image". If the pointer to a double "minval" is not NULL, the minimum value is stored in the space pointed to by "minval", the maximum value is stored in the space pointed to by "maxval" (if not NULL).

The range is also stored in a PIX_INFO structure, attached to the IMAGE structure of image "image" (see AddImageInfo()). In this structure, the range of the image is stored together with an "operation-counter", a value that indicates for which values of the image's "operation-counter" the range was valid. The range information in stored with the name "SI_PixRange".

If get_pixel_range() finds valid range information for an image in the PIX_INFO structure, these values are returned instead of calculating them from the image data.

### *STRUCTURES*

```
typedef struct image_pix_info {
      double minval;
      double maxval;
      int range_op_cnt;
} PIX_INFO;
```

### *RETURN VALUES*

IE_OK (1) on success
Negative error status on failure (see im_error.h)

### *SEE ALSO*

AddImageInfo  post_op

## *get_pixel_size*

***NAME***

get_pixel_size - return memory size of one pixel (in bits)

***SYNOPSIS***

```
#include "im_proto.h"

int get_pixel_size(IMAGE *im)
```

***DESCRIPTION***

get_pixel_size() returns the amount of memory that one pixel in image "im" occupies. This size is specified in bits.

***RETURN VALUES***

The number of bits per pixel on success.
Negative error status on failure (see im_error.h)

---

## *get_sizes*

***NAME***

get_sizes - obtain sizes of an image in the image-silo

***SYNOPSIS***

```
#include "silo.h"

int get_sizes(SILOPTR siloptr, int silo_key, int *sizex, int *sizey)
```

***DESCRIPTION***

| | | |
|---|---|---|
| siloptr | - | Pointer to the image-silo. |
| silo_key | - | Numerical entry label. |
| sizex | - | Variable to return the width of the entry. |
| sizey | - | Variable to return the height of the entry. |

Finds out what the width and height are of a given entry. Returns these values in the variables sizex and sizey.

***RETURN VALUES***

IE_OK (1) on success
IE_NOT_OK (0) if entry was empty

---

## *get_slice*

## *put_slice*

***NAME***

get_slice - copy a 2D-slice from a 3D image

put_slice - copy a 2D-slice into a 3D image

***SYNOPSIS***

```
#include "im_proto.h"

int get_slice(IMAGE *im3d, IMAGE *im2d, int orientation, int slice)

int put_slice(IMAGE *im2d, IMAGE *im3d, int orientation, int slice)
```

***DESCRIPTION***

get_slice() copies the pixel values from the selected slice of the 3D image "im3d" to the 2D image "im2d". The possible "orientation"s are : xy(0), xz(1) or yz(2). "im3d" must be a 3D image, "im2d" is adjusted to the sizes of the slice that is copied. "slice" must be in the range <0..(len-1)>, with len the total number of slices in the 3D image for the selected direction.

put_slice() copies the pixel values from the 2D image "im2d" to the selected slice of the 3D image "im3d". The  possible "orientation"s are : xy(0), xz(1) or yz(2). "im2d" must be a 2D image, "im3d" is adjusted to 3D. "slice" must be in the range <0..(len?-1)>, with len? the total number of slices in the 3D image for the selected direction.

***RETURN VALUES***

IE_OK (1) on succes
Negative error status on failure (see im_error.h)

## *get_super_im*

## *get_super_clut*

## *get_super_histo*

***NAME***

get_super_im - retrieve pointer to super image object.

get_super_clut - retrieve pointer to super clut object.

get_super_histo - retrieve pointer to super histogram object.

***SYNOPSIS***

```
#include "im_infra.h"

void *get_super_im(void)

void *get_super_clut(void)

void *get_super_histo(void)
```

***DESCRIPTION***

get_super_im() returns a pointer to the global super image object. Through this object all creations and destructions of images are published.

get_super_clut() returns a pointer to the global super clut object. Through this object all changes to cluts are published.

get_super_histo() returns a pointer to the global super histogram object. Through this object all creations and destructions of histograms are published.

***RETURN VALUES***

A pointer to the requested object.

***SEE ALSO***

spb_publish  spb_subscribe  spb_unsubscribe

## *getc*

## *getchar*

## *fgetc*

## *getw*

***NAME***

getc, getchar, fgetc, getw - get character or word from stream

***SYNOPSIS***

```
#include <stdio.h>

int getc(FILE *stream)

int getchar(void)

int fgetc(FILE *stream)

int getw(FILE *stream)
```

***DESCRIPTION***

These functions are interface functions to the standard C functions as implemented on the current operating system. The functionality of these functions is:

getc() returns the next character from the named input stream.

getchar() is identical to getc(stdin).

fgetc() behaves like getc(), but is a genuine function, not a macro.

getw() returns the next word from the named input stream. It returns the constant EOF upon end of file or error, but since that is a good integer value, feof() and ferror() should be used to check the success of getw(). getw() assumes no special alignment in the file.

***RETURN VALUES***

These functions return the integer constant EOF at end of file or upon read error.

A stop with message, "Reading bad file", means an attempt has been made to read from a stream that has not been opened for reading by fopen.

***SEE ALSO***

fopen  putc  gets  scanf  fread  ungetc

## *GETENV*

## *SETENV*

***NAME***

GETENV, SETENV - retrieval and storage of environment variables

***SYNOPSIS***

```
#include "support.h"

void SETENV(const char *string)

char *GETENV(const char *name)
```

***DESCRIPTION***

SETENV() puts the variable specified in "string" into the environment of the program. If the operating system also supports an environment, the variable is also put in the system's environment. The syntax of "string" is "variable=value". If "string" is of the form "variable" or "variable=" the variable is cleared (not destroyed). Spaces before of after variable and value are  stripped, however spaces within the value are kept. If the equal sign "=" is not supplied, the first white space after variable is considered to be the separator. If variable already exists in the program's environment the space of the existing value is freed using free().

GETENV() retrieves the value of the environment variable "name". First the environment of the program is searched and if "name" is not found, the operating system's environment is searched (if present). When neither the program's nor the system's environment contain the variable, NULL is returned

***RETURN VALUES***

SETENV() returns nothing.
GETENV() returns a pointer to the value of the variable or NULL if not found

## *getlogin*

***NAME***

getlogin - get login name

***PLATFORM***

UNIX.

***SYNOPSIS***

```
char *getlogin(void)
```

***DESCRIPTION***

This function is an interface to the standard C function as implemented on the current system. The functionality of this function is:

getlogin() returns a pointer to the login name as found in /etc/utmp. It may be used in conjunction with getpwnam() to locate the correct password file entry when the same userid is shared by several login names.

***RETURN VALUES***

Returns NULL (0) if name not found.

## *gets*

## *fgets*

***NAME***

gets, fgets - get a string from a stream

***SYNOPSIS***

```
#include <stdio.h>

char *gets(char *s)

char *fgets(char *s, int n, FILE *stream)
```

***DESCRIPTION***

These functions are interface functions to the standard C functions as implemented on the current operating system. The functionality of these functions is:

gets() reads a string into "s" from the standard input stream stdin. The string is terminated by a newline character, which is replaced in "s" by a null character. gets() returns its argument.

fgets() reads n-1 characters, or up to a newline character, whichever comes first, from the stream into the string "s". The last character read into "s" is followed by a null character. fgets() returns its first argument.

***RETURN VALUES***

gets() and fgets() return the constant pointer NULL upon end of file or error.

***SEE ALSO***

puts  getc  scanf  fread  ferror

## *glc_entropy*

## *glc_contrast*

## *glc_asymmetry*

***NAME***

glc_entropy - texture measure, co-occurence of grey-levels

glc_contrast - texture measure, co-occurence of grey-levels

glc_asymmetry - texture measure, co-occurence of grey-levels

***SYNOPSIS***

```
#include "im_proto.h"

double glc_entropy(IMAGE *input, IMAGE *mask, int vectorx, int
vectory)

double glc_contrast(IMAGE *input, IMAGE *mask, int vectorx, int
vectory)

double glc_asymmetry(IMAGE *input, IMAGE *mask, int vectorx, int
vectory)
```

***DESCRIPTION***

The functions calculate a 2-dimensional histogram of the combinations of grey-values of pixels that are the startpoint/endpoint of a vector with a specified ("vectorx","vectory") displacement.

The functions calculate:

| | |
|---|---|
| glc_asymmetry | Asymmetry of the histogram<br>Sum over g1,g2 of (p(g1,g2)**2) |
| glc_contrast | Contrast of the histogram<br>Sum over g1,g2 of ((g1-g2)**2)*p(g1,g2) |
| glc_entropy | Entropy of the histogram<br>Sum over g1,g2 of p(g1,g2)*log(p(g1,g2)) |

Where g1 and g2 are the grey-values at the start and end of the vector, and p(g1,g2) is the chance of this combination in the image.

The calculation of the texture is only done in the areas where the bit-image "mask" has value 1.

***RETURN VALUES***

The texture value is returned. In case of error, this is 0.

***SEE ALSO***

box_dimension  gld_mean  gld_entropy gld_contrast  gld_asymmetry

glr_nonuniformity  glr_shortrunemphasis  glr_longrunemphasis
glr_greynonuniformity  glr_percentage  edge_average  dist_average

## *gld_mean*

## *gld_entropy*

## *gld_contrast*

## *gld_asymmetry*

***NAME***

gld_mean - texture measure, difference of grey-levels

gld_entropy - texture measure, difference of grey-levels

gld_contrast - texture measure, difference of grey-levels

gld_asymmetry - texture measure, difference of grey-levels

***SYNOPSIS***

```
#include "im_proto.h"

double gld_mean(IMAGE *input, IMAGE *mask, int vectorx, int vectory)

double gld_entropy(IMAGE *input, IMAGE *mask, int vectorx, int
vectory)

double gld_contrast(IMAGE *input, IMAGE *mask, int vectorx, int
vectory)

double gld_asymmetry(IMAGE *input, IMAGE *mask, int vectorx, int
vectory)
```

***DESCRIPTION***

The functions calculate a histogram of the absolute differences of all combinations of pixels that are the startpoint/endpoint of a vector with a specified ("vectorx","vectory") displacement.

The functions calculate:

| | |
|---|---|
| gld_mean | Mean of the histogram<br>Sum of i*p(i) |
| gld_entropy | Entropy of the histogram<br>Sum of p(i)*log(p(i)) |
| gld_contrast | Contrast of the histogram<br>Sum of (i**2)*p(i) |
| gld_asymmetry | Asymmetry of the histogram<br>Sum of (p(i)**2) |

Where i is the absolute difference, and p(i) is the chance of that absolute difference in the image.

The calculation of the texture is only done in the areas where the bit-image "mask" has value 1.

***RETURN VALUES***

The texture value is returned. In case of error, this is 0.

***SEE ALSO***

box_dimension  glc_entropy  glc_contrast  glc_asymmetry  glr_nonuniformity  glr_shortrunemphasis  glr_longrunemphasis  glr_greynonuniformity  glr_percentage  edge_average  dist_average

## *glr_nonuniformity*

## *glr_shortrunemphasis*

## *glr_longrunemphasis*

## *glr_greynonuniformity*

## *glr_percentage*

***NAME***

glr_nonuniformity - texture measure, run-length statistics

glr_shortrunemphasis - texture measure, run-length statistics

glr_longrunemphasis - texture measure, run-length statistics

glr_greynonuniformity - texture measure, run-length statistics

glr_percentage - texture measure, run-length statistics

***SYNOPSIS***

```
#include "im_proto.h"

double glr_nonuniformity(IMAGE *input, IMAGE *mask)

double glr_shortrunemphasis(IMAGE *input, IMAGE *mask)

double glr_longrunemphasis(IMAGE *input, IMAGE *mask)

double glr_greynonuniformity(IMAGE *input, IMAGE *mask)

double glr_percentage(IMAGE *input, IMAGE *mask)
```

***DESCRIPTION***

The functions calculate a histogram of the grey-value/run-length combinations in the image.

The functions calculate:

| | |
|---|---|
| glr_shortrunemphasis | Run-length short run emphasis<br>Sum over i,j of p(i,j)/(j**2) |
| glr_longrunemphasis | Run-length long run emphasis<br>Sum over i,j of p(i,j)*(j**2) |
| glr_greynonuniformity | Run-length grey-level non-uniformity<br>Sqrt (Sum over i of ((Sum over j of p(i,j))**2)) |
| glr_nonuniformity | Run-length non-uniformity<br>Sqrt (Sum over j of ((Sum over i of p(i,j))**2)) |
| glr_percentage | Run-length percentage |

100*(Number of runs/number of pixels)

Where i is the grey-level, j is the run-length and p(i) is the chance of that combination in the image.

The calculation of the texture is only done in the areas where the bit-image "mask" has value 1.

***RETURN VALUES***

The texture value is returned. In case of error, this is 0.

***SEE ALSO***

box_dimension  gld_mean  gld_entropy gld_contrast  gld_asymmetry
glc_entropy  glc_contrast  glc_asymmetry  edge_average  dist_average

---

## *gravx*

***NAME***

gravx - obtain x coordinate of center of gravity of object

***SYNOPSIS***

```
#include "im_aio.h"

double gravx(LIST *link)
```

***DESCRIPTION***

| | | |
|---|---|---|
| link | - | Link pointing to object |

AIO primitive to obtain value of an object feature

gravx() returns the x coordinate of the center of gravity of the object pointed to by "link" if this has previously been measured.

***RETURN VALUES***

x coordinate of center of gravity of object on success
0.0 if link is not an object or if center of gravity has not been measured

***SEE ALSO***

measure  object_shape_meas  object_dens_meas

---

## *gravy*

***NAME***

gravy - obtain y coordinate of center of gravity of object

***SYNOPSIS***

```
#include "im_aio.h"

double gravy(LIST *link)
```

***DESCRIPTION***

| | | |
|---|---|---|
| link | - | Link pointing to object |

AIO primitive to obtain value of an object feature

gravy() returns the y coordinate of the center of gravity of the object pointed to by "link" if this has previously been measured.

***RETURN VALUES***

y coordinate of center of gravity of object
0.0 if link is not an object or if center of gravity has not been measured

***SEE ALSO***

measure  object_shape_meas  object_dens_meas

## *greater0_ok*

***NAME***

greater0_ok - check if a integer value is greater than zero

***SYNOPSIS***

```
#include "im_infra.h"

int greater0_ok(int value, char *text)
```

***DESCRIPTION***

The integer value "value" is checked to see if it is greater than zero or not. If it is zero or negative an error is generated and the following message is added to the error-stack:

```
      <text> [<value>] must be bigger than 0
```

***NOTE***

This function can only handle integer values, to check on float values, use the function fgreater0_ok().

***RETURN VALUES***

IE_OK (1) if the value is bigger than zero
IE_NOT_OK (0) if it is zero or negative

***SEE ALSO***

fgreater0_ok  positive_ok

## *greduce*

***NAME***

greduce - grey value reduction of image

***SYNOPSIS***

```
#include "im_proto.h"

int greduce(IMAGE *in, IMAGE *out, int nlev, int auto_contr)
```

***DESCRIPTION***

greduce() reduces the number of grey values in image "in" to "nlev" levels and stores the result in image "out". By default the range of the pixel values is assumed to be 0 - 255. Any values outside that range are clipped to 0 and 255. The "auto_contr" parameter determines that the actual range of the pixel values should be calculated and used instead of the 0 - 255 range.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

dither  pseudo

---

## *grey_dilation*

***NAME***

grey_dilation - grey value dilation (local maximum)

***SYNOPSIS***

```
#include "im_proto.h"

int grey_dilation(IMAGE *in, IMAGE *out, int filtx, int filty, int
filtz)
```

***DESCRIPTION***

Non-linear local maximum filter.
Image "in" is scanned with a moving window with sizes "filtx" and "filty" and "filtz" (for 3D images). For each window position the maximum of the pixel values within the window is calculated. This local maximum value is stored in the pixel in image "out" that corresponds with the central pixel within the window

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

grey_erosion

---

## *grey_erosion*

***NAME***

grey_erosion - grey value erosion (local minimum)

***SYNOPSIS***

```
#include "im_proto.h"

int grey_erosion(IMAGE *in, IMAGE *out, int filtx, int filty, int
filtz)
```

***DESCRIPTION***

Non-linear local minimum filter.
Image "in" is scanned with a moving window with sizes "filtx" and "filty" and "filtz" (for 3D images). For each window position the minimum of the pixel values within the window is calculated. This local minimum value is stored in the pixel in image "out" that corresponds with the central pixel within the window

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

grey_dilation

## *grey_mean*

## *trans_mean*

## *od_mean*

***NAME***

grey_mean - mean of grey values of objects area

trans_mean - mean of transmission over objects area

od_mean - mean of optical density over objects area

***SYNOPSIS***

```
#include "im_aio.h"

double grey_mean(LIST *link)

double trans_mean(LIST *link)

double od_mean(LIST *link)
```

***DESCRIPTION***

link - Link pointing to the object

AIO primitives to obtain value of an object feature

grey_mean() returns the mean of the pixel values of the object pointed to by "link".

trans_mean() returns the mean of the pixel values of the object pointed to by "link" according to a transmission pixel table.

od_mean() returns the mean of the pixel values of the object pointed to by "link" according to a optical density pixel table

The features must have been measured beforehand with object_dens_meas().

***RETURN VALUES***

Mean of all pixel values of object on success.
0.0 if link is not an object or if mean value has not been measured

***SEE ALSO***

measure object_shape_meas object_dens_meas list_label

## *grey_morph_round*

## *grey_morph_ellipse*

## *grey_morph_hollow_ellipse*

## *grey_morph_diamond*

## *grey_morph_arbit*

***NAME***

grey_morph_round, grey_morph_ellipse, grey_morph_hollow_ellipse, grey_morph_diamond, grey_morph_arbit - grey valued morphology

***SYNOPSIS***

```
#include "im_proto.h"

int grey_morph_round(IMAGE *in, IMAGE *out, int fsize, int norm, int
type)

int grey_morph_ellipse(IMAGE *in, IMAGE *out, int x_axis, int y_axis,
double orient, int norm, int type)

int grey_morph_hollow_ellipse(IMAGE *in, IMAGE *out, int x_axis, int
y_axis, double orient, int conn, int type)

int grey_morph_diamond(IMAGE *in, IMAGE *out, int fsize_nw_se, int
fsize_ne_sw, int type)

int grey_morph_arbit(IMAGE *in, IMAGE *filter, IMAGE *out, int norm,
int type)
```

***DESCRIPTION***

These functions are morphological operators with different shaped structuring elements: circular, elliptic, hollow elliptic, diamond and arbitrary shaped.

Each of these functions can perform a dilation, an erosion, an opening or a closing on GREY_2D images. The round, elliptic and hollow elliptic also support a uniform and a kuwahara filter. The arbitrary shaped supports the uniform filter too.

For FLOAT_2D images, the round, the elliptic and the hollow elliptic shapes support dilation, erosion, opening, closing and uniform.

Kuwahara, diamond shaped and arbitrary shaped are not supported for FLOAT_2D.

For all functions the image "in" is the input image and the image "out" is the output image. "type" is the type of operator, being :

| | |
|---|---|
| UNIF (1) | uniform |
| DILA (2) | dilation |
| EROS (3) | erosion |
| CLOSE(4) | closing |
| OPEN (5) | opening |

|  |  |
|---|---|
| KUWA (6) | kuwahara |

If "norm" is set (1), the result of the uniform filter is normalized.

The size of the structuring element of the grey_morph_round() function is specified by "fsize", the diameter of the circle.

The sizes of the elliptic structuring element of grey_morph_ellipse() and grey_morph_hollow_ellipse() are given by "x_axis" and "y_axis", the axis of the ellipse by an orientation "orient" of 0 degrees.

The sizes of the diamond shaped structuring element of grey_morph_diamond() are specified by "fsize_nw_se" and "fsize_ne_sw".

For the arbitrary shaped structuring element, a separate image is needed in which the structuring element is present. The sizes of this structuring element should be odd in both dimensions.

The sizes of all the types of structuring elements should be odd.

***LITERATURE***

J. Serra, Image Analysis and Mathematical Morphology, Academic Press.

B.J.H. Verwer, L.J. van Vliet and P.W. Verbeek, Binary and Grey-value Skeletons Metrics and Algorithms, International Journal of Pattern Recognition and Artificial Intelligence, Vol. 7(5), 1993.

R. v.d. Boomgaard, Mathematical Morphology: Extensions towards Computer Vision, Ph.D.-thesis, University of Amsterdam, 1992.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

upper_gskeleton  lower_gskeleton  uniform_round  kuwahara_round

## *grey_stdev*

## *trans_stdev*

## *od_stdev*

***NAME***

grey_stdev - standard deviation of grey values of objects area

trans_stdev - standard deviation of transmission over objects area

od_stdev - standard deviation of optical density over objects area

***SYNOPSIS***

```
#include "im_aio.h"

double grey_stdev(LIST *link)

double trans_stdev(LIST *link)

double od_stdev(LIST *link)
```

***DESCRIPTION***

| | | |
|---|---|---|
| link | - | Link pointing to the object |

AIO primitives to obtain value of an object feature

grey_stdev() returns the standard deviation of the pixel values of the object pointed to by "link".

trans_stdev() returns the standard deviation of the pixel values of the object pointed to by "link" according to a transmission pixel table.

od_stdev() returns the standard deviation of the pixel values of the object pointed to by "link" according to a optical density pixel table.

The features must have been measured beforehand with object_dens_meas().

***RETURN VALUES***

Standard deviation of pixel values of object on success.
0.0 if link is not an object or if standard deviation has not been measured

***SEE ALSO***

measure  object_shape_meas  object_dens_meas  list_label

## *grey_sum*

## *trans_sum*

## *od_sum*

***NAME***

grey_sum - sum of grey values of objects area

trans_sum - sum of transmission over objects area

od_sum - sum of optical density over objects area

***SYNOPSIS***

```
#include "im_aio.h"

double grey_sum(LIST *link)

double trans_sum(LIST *link);

double od_sum(LIST *link)
```

***DESCRIPTION***

| | | |
|---|---|---|
| link | - | Link pointing to the object |

AIO primitives to obtain value of an object feature

grey_sum() returns the sum of the pixel values of the object pointed to by "link".

trans_sum() returns the sum of the pixel values of the object pointed to by "link" according to a transmission pixel table.

od_sum() returns the sum of the pixel values of the object pointed to by "link" according to a optical density pixel table.

The features must have been measured beforehand with object_dens_meas().

***RETURN VALUES***

Sum of all pixel values of object.
0.0 if link is not an object or if sum of pixels has not been measured

***SEE ALSO***

measure  object_shape_meas  object_dens_meas  list_label

## *handle_pim*

***NAME***

handle_pim - handle the point image display buffer

***PLATFORM***

UNIX, Macintosh.

***SYNOPSIS***

```
#include "disp_p.h"

int handle_pim(int activate)
```

***DESCRIPTION***

handle_pim() shows ("activate" = 1) or hides ("activate" = 0) the pim_window, the window in which the pixel information is shown when clicking the left mouse button in an image display window. For more information see point_im_display_buf() or the demo "my_point.c" in the standard demo directory.

***RETURN VALUES***

None

***SEE ALSO***

point_im_display_buf  point_im

## *have_diff*

***NAME***

have_diff - Lee-Haralick-Verbeek edge detector

***SYNOPSIS***

```
#include "im_proto.h"

int have_diff(IMAGE *in, IMAGE *out, int mode)
```

***DESCRIPTION***

Differential edge detection based upon local minimum and local maximum filters. Within the moving window, with dimensions 3*3, the minimum and the maximum are calculated. For both these values the absolute differences with the central pixel in the window are obtained.
The smaller of these differences becomes the output pixel value and is stored in the corresponding pixel in image "out".
This value is used as a sharp edge detector and is called the Lee-Haralick variant.
A variant of the algorithm modifies the final result in such a way that the result is multiplied by a factor -1 is the difference with the local minimum was chosen, thus taking into account the original sign of the difference. This signed variant can be used for edge sharpening and is called the Verbeek variant. If the output image of the Verbeek variant is added to the original image, the result is to replace each pixel of the original image by either the local minimum or the local maximum value (taken in a 3*3 neighborhood), depending upon which value is closer to the original pixel value. The result may be considered as a non-linear way of edge sharpening.

| | | |
|---|---|---|
| "mode" | 1 | Lee-Haralick variant |
| | 0 | Verbeek variant. |

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

## *height*

***NAME***

height - height of object

***SYNOPSIS***

```
#include "im_aio.h"

int height(LIST *link)
```

***DESCRIPTION***

| | | |
|---|---|---|
| link | - | Link pointing to the object |

AIO primitive to obtain value of an object feature

height() returns the height of the object pointed to by "link".

This feature need not be specified beforehand as it is automatically measured during the labeling process.

***RETURN VALUES***

The height of the object in pixels on success
0 if link is not an object

***SEE ALSO***

measure  object_shape_meas  object_dens_meas  list_label

## *help*

***NAME***

help, ? - get help information

***SYNOPSIS***

```
help <pattern>

? <pattern>
```

***DESCRIPTION***

The help command (or ?) supplies information on the specified topic <pattern>. If a command or function name is specified, information on that command (function) is given. If wildcards are present in <pattern>, all commands and/or variables that match that pattern are listed. If a double question mark (??) is followed by a pattern than all help files on all commands/functions that match that pattern are displayed.

HELP OPTIONS:

| | |
|---|---|
| ? | This information |
| help | This information |
| ?command | Information on SCIL command |
| ?pattern | List of variables/functions matching pattern |
| ?strcmp | Information about strcmp() |
| ?count | Information about global variable "count" |

Pattern may contain the wildcard "*" to match anything

***EXAMPLE***

```
[C1] ?s*p
sleep
strcmp
strncmp
[C2]
```

## hide_object

## hide_object_at

***NAME***

hide_object - obscure labeled object

hide_object_at - obscure labeled object at specified position

***SYNOPSIS***

```
#include "im_aio.h"

int hide_object(IMAGE *image, LIST *link)

int hide_object_at(IMAGE *image, LIST *link, int x, int y)
```

***DESCRIPTION***

| | | |
|---|---|---|
| image | - | Pointer to image with labeled objects |
| link | - | Link pointing to object |
| x, y | - | coordinate of top left of objects rectangle |

hide_object() obscures the object in a labeled image by setting all its pixels at zero.

hide_object_at() obscures the object in a labeled image if the left top of the objects surrounding rectangle is at position "x", "y".

***NOTE***

The object is not removed from the list. You need to call rm_object() followed by update() to remove an object from the list.

***EXAMPLE***

```
To hide objects touching the edge of an image:

#include "image.h "
#include "im_aio.h"
LIST *l, *o;

readfile("cermet",a,0,0);
threshold(a,b,128);
invert_im(b,b);
l = list_label(b,c,8,0);
FORALL(o,l) if(edge_object(c,o)) hide_object(c,o);
display_image(c);
l = rm_list(l);
```

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

## hild_skelet

### NAME

hild_skelet - skeletonization according to Hilditch

### SYNOPSIS

```
#include "im_proto.h"

int hild_skelet(IMAGE *in, IMAGE *out, int iter, int endp, int bound)
```

### DESCRIPTION

Changes the objects in image "in" into skeletons and stores the result in image "out". The skeleton is defined as a set of connected, one pixel thick arcs, lying midway between the object boundaries and being a topological retraction with the same connectedness as the original object. The skeleton represents the morphologic ("shape") features of the original object.

The thinning operation may be executed for only a limited number of cycles, as specified by the parameter "iter". Full skeletonization results if the value specified for this parameter is equal or larger than half the image-size.

"endp" specifies that the endpixels of the skeleton must be preserved with each thinning iteration (1 is preserve, 0 is do not preserve).

"bound" specifies that the edge around the image must be set to foreground (1) or to background (0) pixels.

### RETURN VALUES

IE_OK (1) on success  
Negative error status on failure (see im_error.h)

### SEE ALSO

holt_skelet  skelpoints

## *hist*

## *:*

***NAME***

hist - display the commands history

: - recall command

***PLATFORM***

UNIX.

***SYNOPSIS***

```
hist [start], [end]

:[number]

:[pattern]
```

***DESCRIPTION***

The history command "hist" shows previously given commands. The numbers "start"" and "end" specify which ones should be listed. If no "number" is given, the last 20 commands are listed. If a single "number" is specified, only that particular command is shown. A range can be specified by supplying the begin and end separated by a comma. If the argument before the comma is omitted, "start" is 1 (the first commands). If the "number" after the comma is omitted, "end" is the  last command.

Apart from recalling commands by their number, also a "pattern" can be used. This "pattern" must be the start of the command line to recall. E.g. ":read" recalls the last command that started with "read".

Typing a colon ":" followed by a command number or pattern recalls this command. In case of a pattern only the last 20 commands are compared.

A recalled command is placed in the insert mode of the line editor.

***EXAMPLE***

```
hist                list the last 20 commands
hist 3              show command number 3
hist 3,             list command 3 up to last command
hist  ,30           list command 1 up to 30
hist 3,30           show command 3 up to 30

:<number>           recall command [C<number>]
:<pattern>          recall last command starting with <pattern>
:                   recall last given command.
```

## *hist2d*

***NAME***

hist2d - 2-dimensional image histogram calculation

***SYNOPSIS***

```
#include "im_proto.h"

int hist2d(IMAGE *in1, IMAGE *in2, IMAGE *out, int clip)
```

***DESCRIPTION***

Calculate the histogram of co-incidences of grey values in the image "in1" and "in2" and store the resulting two-dimensional histogram in image "out". For each element of the image "out" with indices I and J, the number of times a pixel in "in1" has the value I and its corresponding pixel in "in2" has the value J is counted and stored as the new pixel value in the output image "out". When "clip" is set (=1), pixel values in the input images "in1" and "in2" that are either negative or greater than the size of the output image, are truncated to zero or the appropriate image size of "out" respectively. If "clip" is not set (=0), pixel values outside the range from zero to the output image size, cause an error condition..

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

histdata

## *histdata*

***NAME***

histdata - image histogram calculation

***SYNOPSIS***

```
#include "im_proto.h"

int histdata(IMAGE *in, VAR_OBJECT *data, int len, int clip)
```

***DESCRIPTION***

Calculates the grey level histogram of image "in" and stores the resulting histogram in the var_object "data". For each pixel value in image "in" the number of pixels is counted in a corresponding element of array "data", a so-called bin. "len" is the number of bins. When "clip" is true (=1), pixel values in image "in" that are less than zero or greater than "len" are truncated to zero and "len" respectively. If "clip" is false (=0), pixel values outside the range zero to "len" cause an error condition.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

hist2d

---

## *histogram*

***NAME***

histogram - plot histogram in display window

***SYNOPSIS***

```
#include "im2scil.h"

int histogram(IMAGE *in)
```

***DESCRIPTION***

Calculates the grey level histogram of image "in" and plots the resulting histogram.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

histdata  hist2d  plot_histogram

---

## *histogram_by_name*

## *histogram_ok*

## *is_histogram*

***NAME***

histogram_by_name - get the pointer to a histogram by its name

histogram_ok - check if pointer is a histogram and pop alert if not

is_histogram - check if pointer is a histogram object

***SYNOPSIS***

```
#include "im_infra.h"

HISTOGRAM *histogram_by_name(char *name, int case_check)

int histogram_ok(HISTOGRAM *histo)

int is_histogram(HISTOGRAM *histo)
```

***DESCRIPTION***

histogram_by_name() retrieves the pointer to the histogram "name". "case_check" specifies whether a distinction between lower case and upper case characters should to be made. If zero then no distinction is made.

histogram_ok() and is_histogram() check the pointer "histo" to see if it points to a valid histogram. The linked list in which all the histograms are present is scanned for the occurrence of "histo". If it is a valid histogram, a true status (1) is returned, else a false status (0). Additionally, if it is not a valid histogram pointer, histogram_ok() generates an error and adds the following message to the error-stack:

```
Non existing histogram pointer.
```

***SEE ALSO***

create_histogram  destroy_histogram  histo_data  copy_histogram
histogram_to_image  histogram_to_var_object  image_to_histogram
histogram_comment  dump_histogram  list_histograms  show_histogram_info

## *histogram_comment*

## *dump_histogram*

## *list_histograms*

## *show_histogram_info*

***NAME***

histogram_comment - attach a comment string to a histogram object

dump_histogram - show the histogram data in ASCII

list_histograms - list all histograms

show_histogram_info - list information on a histogram

***SYNOPSIS***

```
#include "im_infra.h"

int histogram_comment(HISTOGRAM *histo, char *comment)

int dump_histogram(HISTOGRAM *histo, char *file, int num)

int list_histograms(void)

int show_histogram_info(HISTOGRAM *histo)
```

***DESCRIPTION***

histogram_comment() adds a (null-terminated) string "comment" to the structure of the histogram "histo". The string may be of any length as long as it is null-terminated. The function itself allocates memory for the string, so if adding comment to an histogram while not using this function, be sure that the memory was allocated with malloc() for other functions rely on it (they use free()). Any previously attached comment is removed before the comment is added.

dump_histogram() dump the data of histogram "histo" in ASCII to either the controlling terminal or a file. If a name is specified for the file ("file") then the data will be stored in a file, else the data will be dumped on the terminal. The last parameter "num" specifies the number of values that will be printed on a single line (default = 1). If the data is dumped to an already existing file, the old file will be overwritten without warning (provided the file permissions allow this).

list_histograms() displays a list of all existing histogram on the controlling terminal.

show_histogram_info() displays information about a histogram object on the terminal. The name, sizes, lowest and highest bin median values and the comment are shown on the controlling terminal.

***SEE ALSO***

create_histogram  destroy_histogram  histo_data  copy_histogram

histogram_to_image histogram_to_var_object image_to_histogram
histogram_by_name histogram_ok is_histogram

---

## ***histogram_to_image***

## ***image_to_histogram***

## ***histogram_to_var_object***

***NAME***

histogram_to_image - copy the histogram data to an image

image_to_histogram - copy image data to an histogram

histogram_to_var_object - copy the histogram data to an var_object

***SYNOPSIS***

```
#include "im_infra.h"

int histogram_to_image(HISTOGRAM *histo, IMAGE *image, int out_type)

int image_to_histogram(IMAGE *image, HISTOGRAM *histo)

int histogram_to_var_object(HISTOGRAM *histo, VAR_OBJECT *object)
```

***DESCRIPTION***

histogram_to_image() copies the data of histogram "histo" to the image "image". The image sizes are changed to match the sizes of the histogram. The image type is not changed.

image_to_histogram() copies the data of image "image" to the histogram object "histo". The sizes of the histogram are changed to match those of the image.

histogram_to_var_object() copies the data of histogram "histo" to the var_object "object". The sizes of the var_objects are changed to match those of the histogram. The type of the var_object is changed to LONG_T.

***SEE ALSO***

create_histogram destroy_histogram histo_data copy_histogram
histogram_by_name histogram_ok is_histogram
histogram_comment dump_histogram list_histograms show_histogram_info

---

## *hit_or_miss*

***NAME***

hit_or_miss - "hit or miss" transform

***SYNOPSIS***

```
#include "im_proto.h"

int hit_or_miss(IMAGE *in, IMAGE *out, IMAGE *se, int bound)
```

***DESCRIPTION***

Performs a "hit or miss" transform on image "in" using the "hit or miss" mask encoded in image "se" and stores the result in image "out". The "hit or miss transform" calculates the intersection of the erosion of image "in" using structuring element S1 and the erosion of the complement (inverted image) of image "in" using structuring element S2. The pixels belonging to S1 are encoded in image "se" with positive grey values, those belonging to S2 are encoded with negative grey values.

"bound" specifies that the edge around the image must be set to foreground (1) or to background (0) pixels.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

arbit_erosion  arbit_dilation  real_time_recognizer  t_morphology

## ***holt_skelet***

***NAME***

holt_skelet - skeleton according to Holt, Stewart, Clint & Perrott

***SYNOPSIS***

```
#include "im_proto.h"

int holt_skelet(IMAGE *in, IMAGE *out, int iter, int bound)
```

***DESCRIPTION***

Changes the objects in image "in" into skeletons and stores the result in image "out". The skeleton is defined as a set of connected, one pixel thick arcs, lying midway between the object boundaries and being a topological retraction with the same connectedness as the original object. The skeleton represents the morphologic ("shape") features of the original object.

The thinning operation may be executed for only a limited number of cycles, as specified by the parameter "iter". Full skeletonization results if the value specified for this parameter is equal or larger than half the image-size.

"bound" specifies that the edge around the image must be set to foreground (1) or to background (0) pixels.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

hild_skelet  skelpoints

## *homomorphic*

***NAME***

homomorphic - contrast enhancement filter

***SYNOPSIS***

```
#include "im_proto.h"

int homomorphic(IMAGE *in IMAGE *,out, double low_amplitude, double
filt_size)
```

***DESCRIPTION***

This FFT based function can only be used on 2D images of which sizes are a power of two. The filter is based on the idea that the image is the result of multiplication of an illumination function with a "scene" function. The illumination function is assumed to be composed of low frequency components only. The aim of the filter is to remove the contribution of the illumination function to the image by executing the following steps.

1) Take logarithm of image "in". The result is the SUM of an illumination and a scene component.

2) Apply a high pass filter. The filter size is controlled through "filt_size"; the attenuation of the lower frequency components by "low_amplitude".

3) Reverse of 1) and store the result in the image "out".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

local_contrast

## *hull*

***NAME***

hull - object convex hull detection

***SYNOPSIS***

```
#include "im_proto.h"

int hull(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

Calculate the convex hull of each object in the labeled image "in" and store the result in image "out". For each object in "in", all combinations of two contour points are connected by a straight line. If a background pixel is found on such a line, it is added to the original object. This operation closes all holes in an object. The contour of an object is also smoothed, as all gaps in it are filled.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

label  objectsize  rhull  small_object_removal

## *hypot*

## *cabs*

***NAME***

hypot, cabs - Euclidean distance

***SYNOPSIS***

```
#include <math.h>

double hypot(double x, double y)

double cabs(struct { double x, y;}z)
```

***DESCRIPTION***

These functions are interface functions to the C functions as implemented on the current operating system. The functionality of these functions is:

hypot() and cabs() return

```
      sqrt("x"*"x" + "y"*"y"),
```

taking precautions against unwarranted overflows.

***SEE ALSO***

exp  sqrt

## *Ibenke*

***NAME***

Ibenke - interactive search for texture segmentation filter

***SYNOPSIS***

```
#include "itools.h"

int Ibenke(IMAGE *filter, IMAGE *out, double gain, double
convergence, double sigma, int width, int height)
```

***DESCRIPTION***

Ibenke() asks the user to draw a rectangle in the object and in the background textured regions, and tries to find a (sub-)optimal separation filter. The convergence speed is determined by "gain". The search is finished when the energy difference between the two textures is less than "convergence" compared to the previous iteration. If "convergence" is greater than one, it will be converted to the number of iterations to perform. After training, the input image is segmented by applying convolution(), squaring the image and applying a Gaussian filtering with spatial extend given by "sigma". This gives a measure of local energy. The filter dimensions are given by "width" and "height".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

benke  convolution  mul_im  gauss

## *ics_readfile*

***NAME***

ics_readfile - read an image from a file in ICS format

***SYNOPSIS***

```
#include "im_proto.h"

IMAGE *ics_readfile(char *filename, IMAGE *image, ICS *ics_header,
int xpos, int ypos)
```

***DESCRIPTION***

Read the image "filename" stored in ICS-format into the image "image". "filename" may be specified with or without the file extensions ".ics"/".ids". If "USE_NAME" (a NULL pointer) is specified as the image, a new image is created at position "xpos", "ypos", with the same name as the file. If an image is already present with that name, that image will be used.

Information about the image is stored in the ICS structure "ics_header". If a NULL pointer is passed for "ics_header, the header informatin is not returned to the calling program.

***RETURN VALUES***

The pointer to the image in which the data was put, either an existing image or a newly created one.
NULL on failure

***SEE ALSO***

readfile  tiff_readfile  tcl_readfile  aim_readfile  writefile  ics_writefile

## *ics_writefile*

***NAME***

ics_writefile - write an image to a file in ICS format

***SYNOPSIS***

```
#include "im_proto.h"

int ics_writefile(IMAGE *image, char *filename, ICS *ics_header)
```

***DESCRIPTION***

Stores the image "image" into the file "filename" according to the ICS-format. If a NULL pointer is supplied for "ics_header", one is created internally in ics_writefile() and filled with appropriate default values.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

writefile  tiff_writefile  tcl_writefile  readfile  ics_readfile

---

## *ifft*

***NAME***

ifft

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See fast_fourier

---

## *IGreyMap*

***NAME***

IGreyMap - Interactive grey lookup table editor

***SYNOPSIS***

```
#include "itools.h"

void IGreyMap(IMAGE *image)
```

***DESCRIPTION***

IGreyMap() interactively changes the grey lookup table of the image "image". By dragging the knots that are on the red line, the shape of the lookup table can be changed.

The number of knots used to define the curve can be changed from 2 to 12. The leftmost knot as well as the rightmost knot cannot be dragged from the sides. Knots can be added and removed using the right mouse button (<command> key together with the mouse button on the Mac). To add a knot press the button at the location the new knot is to be positioned. To remove an existing knot press the right mouse button on the knot to be removed.

In the editor window several buttons are available for changing the behavior of the editor:

| | |
|---|---|
| "LINE" | Connect the knots using straight lines. |
| "SPLINE" | Connect the knots using splines. |
| "UPDATE" | Update the image "image" continuously, default is that "image" is updated only if the mouse button is released. |
| "BRIGHTER" | Move all knots upwards |
| "DIMMER" | Move all knots downwards |
| "SHARPER" | Enhance the contrast in the table. |
| "DULLER" | Remove contrast. |
| "LINEAR" | Reset the editor to a linear curve. |
| "OK" | Exit the editor and attach the new lookup table to "image". |
| "CANCEL" | Exit the editor, discarding the new lookup table |
| "HELP" | Short on-line help. |

***RETURN VALUES***

None

***SEE ALSO***

IThreshold

## *im_begin_func*

## *im_end_func*

## *im_report_error*

## *im_get_status*

## *im_debug_stack*

***NAME***

im_begin_func - push a function name on the error-stack

im_end_func - pop a function name from the error-stack

im_report_error - report an error occurrence to the error-stack

im_get_status - retrieve error status from error-stack

im_debug_stack - report inconsistent usage of error-stack

***SYNOPSIS***

```
#include "im_error.h "

void im_begin_func(const char *fname)

void im_end_func(const char *fname)

int im_report_error(const char *fname, int status, const char
*message)

int im_get_status(void)

void im_debug_stack(int flag)
```

***DESCRIPTION***

| | |
|---|---|
| fname | name of the C-function |
| status | error value identifying the error |
| message | string with additional information |
| flag | enable/disable run-time checking of error-stack inconsistencies |

Error reporting mechanism for image processing routines. The function im_begin_func() is put at the beginning of each C-function and im_end_func() is put at every exit of that C-function. These two function keep a function-stack up-to-date with the exact location the program is  at.

When an error is detected by a C-function, it should try to handle and correct it. If it can not correct the error, it must report this error to the error-stack. First it must clean up and then report the error using the im_report_error() function. This report must specify the function name "fname", the error-value "status" and an (optional) message string "message" in which additional textual information can be put. Additionally the function should return the error-value to the higher-level function. im_report_error() also calls im_end_func(), so in case of an error calling im_end_func() is superfluous (but not an error).

im_report_error() also returns its "status" argument so it can be used directly in a "return" statement (see EXAMPLE).

After an error is detected and reported, the error-stack is initialized with the location of the error (the current contents of the function-stack and the error-value and message of the function that reported the error. Only higher-level functions that are on this error-stack can put their error-values and -messages on this stack while handling the error from the lower-level function. The error-stack can hold only one error and  its location. When during the handling of this error another error occurs in a different location, this last error is ignored by the error-stack.

im_get_status() retrieves the error-status of the function that is just below the current function on the error-stack. It should be used when the  lower-level function can not return a error-value through its return-value (e.g. the return-type is a float or a pointer). When no error has been been reported yet or when im_get_status() is called from a function that is not in the "frozen" error-stack, it always returns 1 (no error).

When all higher-level functions have handled the error and reported the error-values and messages to the error-stack, the error is published by the global error-object "im_error_stack". The User Interface then gets notified of the fact that an error has occurred and can present that information to the user/programmer.

im_debug_stack() enables/disables the run-time consistency checking of the function- and error-stack. "flag" is 1 enables and "flag" is 0 disables the checking. Empty function-names, typing errors, forgotten im_end_func() calls all will lead to a corrupt error-stack. When the checking is enabled, this will be reported as soon as it is detected. It is recommended to use  this checking only when testing your new code because the amount of output  can be extensive.

The include file "im_error.h " contains a number of pre-defined error values that are used by Image.

***EXAMPLE***

```
Example of a function that can not allocate a temporary buffer due to
insufficient memory:

#include <stdlib.h>
#include "image.h "
#include "im_error.h "

int my_function( IMAGE *in, IMAGE *out )
{
```

```
            int *ptr, size;

      im_begin_func("my_function");

            ...
            size = ...
            ...
            if ( !(ptr = malloc(size)) )
            return im_report_error("my_function", IE_NOMEM,
                                   "No memory for table");
            ...
            ...
            free(ptr);
            im_end_func("my_function");
            return IE_OK;
      }
```

***RETURN VALUES***

im_report_error() returns its second ("status") argument.
im_get_status() returns the error status of the function that is one level down on the error-stack (if an error has previously been reported).

***SEE ALSO***

im_debug_stack  im_get_func_stack_copy  im_clear_errors  im_clear_func_stack
show_func_stack  show_error_stack

---

## *im_from_silo*

***NAME***

im_from_silo - transfer image from silo to destination image

***SYNOPSIS***

```
#include "image.h"
#include "silo.h"

int im_from_silo(SILOPTR siloptr, int silo_key, IMAGE *dstimage)
```

***DESCRIPTION***

| | | |
|---|---|---|
| siloptr | - | Pointer to an image-silo. |
| silo_key | - | Numerical entry silo_key. |
| dstimage | - | Image pointer |

Copies the image at position "silo_key" from the silo "siloptr" to the image "dstimage". The sizes of the image "dstimage" are set to the sizes of the image from the silo.

***RETURN VALUES***

IE_OK (1) on succes
Negative error status on failure (see im_error.h)

---

## *im_input_func*

## *del_im_input_func*

## *im_exposure_func*

## *del_im_exposure_func*

***NAME***

im_input_func, del_im_input_func, im_exposure_func, del_im_exposure_func - functions for image event handling

***PLATFORM***

UNIX, Macintosh.

***SYNOPSIS***

```
#include "disp_p.h"

int im_input_func(void (*fp)(IMAGE *, int, int, int, IM_EVENT), int
handle_err)

int del_im_input_func(void (*fp)(IMAGE *, int, int, int, IM_EVENT))

int im_exposure_func(void (*fp)(IMAGE *), int handle_err)

int del_im_exposure_func(void (*fp)(IMAGE *))
```

***DESCRIPTION***

These functions can be used to indicate that a program is interested in exposure- and input-events for images.

im_input_func() can be used to specify a function that is to be called when input events for an image occurs. "fp" is a pointer to the function to be called. If "handle_err" is set (1), the specified function is removed from the list when an error occurs (and the program is interrupted).

The function to handle the input events must have the following syntax:

```
void func(IMAGE *imptr, int x, int y, int ch, IM_EVENT but)

imptr;      /* image in which the event occurs */
x, y;       /* position of the pointer */
ch;         /* key pressed (if any) */
but;        /* button state */
```

im_exposure_func() can be used to specify a function that is to be called when exposure events for the images occur. "fp" is a pointer to the function to be called. If "handle_err" is set (1), the specified function is removed from the list when an error occurs (and the program is interrupted).

The function to handle the exposure event must have the following syntax:

```
void func(IMAGE *imptr)
```

```
      imptr;      /* image in which the exposure event occurred */
```

del_im_input_func() deletes the specified function "func" from the list of functions that are called on input events.

del_im_exposure_func() deletes the specified function "func" from the list of functions that are called on exposure events.

***NOTE***

These functions form a new interface to the events in images for application programs. The old interface which consisted of the functions add_applic_exposure_func(), add_applic_win_input_func() and the mandatory names for interpreted event handling functions handle_exposure() and handle_win_input() have become obsolete. The support of those function is no longer guaranteed in future versions of SCIL_Image.

***RETURN VALUES***

IE_OK (1)

## *im_set_output_handler*

***NAME***

im_set_output_handler - intercept textual output.

***SYNOPSIS***

```
#include "imtxtout.h"

IM_OHFUNC im_set_output_handler(IM_OHFUNC funcptr)
```

***DESCRIPTION***

im_set_output_handler() registers a function that is used to handle all textual output that has been "printed" using the image_output() function. An application that uses the Image library may have its own way of presenting textual data and therefore wishes to overrule the default way, im_set_output_handler enables this. "functptr" must be a pointer to a function that has the following function header:

```
void WINAPI funcname( int stream, char *buffer)
```

"stream" is the type of the text as defined with image_output().
"buffer" is a text-buffer containing the text to be shown.

im_set_output_handler() returns the previous output handler so an application is capable of overruling and restoring other handlers.

***NOTE***

We use WINAPI on the MS-Windows platform to provide greater flexibility. On other platforms WINAPI is an empty define (see the include file imtxtout.h)

***RETURN VALUES***

im_set_output_handler() returns the previous output handler.

***SEE ALSO***

image_output

## *im_to_silo*

***NAME***

im_to_silo - add existing image to an image-silo

***SYNOPSIS***

```
#include "image.h"
#include "silo.h"

int im_to_silo(SILOPTR siloptr, int silo_key, IMAGE *image)
```

***DESCRIPTION***

| | | |
|---|---|---|
| siloptr | - | Pointer to the image-silo. |
| silo_key | - | Numerical entry in silo. |
| image | - | Pointer to image. |

Transfers the image "image" to the image-silo "siloptr" at position "silo_key".

***RETURN VALUES***

IE_OK (1) on succes
Negative error status on failure (see im_error.h)

## ***im_val_ok***

***NAME***

im_val_ok - check if value(s) is(are) smaller than the image size(s)

***SYNOPSIS***

```
#include im_infra.h"

int im_val_ok(IMAGE *image, int a1 [, int v1, int a2, int v2, ... ])
```

***DESCRIPTION***

The function performs a series of checks on the values "v1", "v2", ... (depending on the amount specified) to see if they are smaller than or equal to the image size(s). A variable number of argument pairs "aX-vX" (up to 8 pairs) can be checked. Each value "vX" is checked if it is positive and smaller than or equal to the image dimension specified by "aX". Each pair of arguments is checked by calling the function val_check(). Each argument "aX" can have one of the following values:

| | |
|---|---|
| WIDTH (1) | check if the corresponding value "vX" is smaller than or equal to the image width. |
| HEIGHT (2) | check if the corresponding value "vX" is smaller than or equal to the image height. |
| DEPTH (3) | check if the corresponding value "vX" is smaller than or equal to the image depth. |
| END (-1) | if less than 8 pairs of arguments are supplied, the first not used argument "aX" must be END to stop checking. |

***RETURN VALUES***

IE_OK (1) if all the values are positive and smaller than the image size(s) specified
IE_NOT_OK (0) if one of the value is bigger than an image size.

***SEE ALSO***

val_check

## *im1ps*

## *im2ps*

## *im3ps*

## *im4ps*

***NAME***

im1ps - print one image in postscript

im2ps - print two images

im3ps - print three images

im4ps - print four images

***SYNOPSIS***

```
#include "im_proto.h"

void im1ps(IMAGE *im, char *title, double xsize, double ysize, int
fntsize, int border, char *filename)

void im2ps(IMAGE *im1, char *title1, IMAGE *im2, char *title2, double
xsize, double ysize, int fntsize, int border, char *filename)

void im3ps(IMAGE *im1, char *title1, IMAGE *im2, char *title2, IMAGE
*im3, char *title3, double xsize, double ysize, int fntsize, int
border, char *filename)

void im4ps(IMAGE *im1, char *title1, IMAGE *im2, char *title2, IMAGE
*im3, char *title3, IMAGE *im4, char *title4, double xsize, double
ysize, int fntsize, int border, char *filename)
```

***DESCRIPTION***

The functions im1ps(), im2ps(), im3ps() and im4ps() output one or more images in a postscript file. The size of the image can be specified (in centimeters) by "xsize" and "ysize", the title of the image by "title" together with the font-size "fntsize" in which it is printed (in points). If "border" is set (1), a rectangle is drawn around the image. The output is stored in the file "filename".

im1ps() plots one image centered on the page.

im2ps() plots two images aligned vertically (the images are numbered from top to bottom).

im3ps() plots three images aligned vertically.

im4ps() plots 4 images in a 2x2 matrix. The images are numbered as:

```
1  2
3  4
```

***SEE ALSO***

ps_head  ps_image  ps_tail

---

## *image_ok*

***NAME***

image_ok - check if the supplied pointer is a valid image pointer

***SYNOPSIS***

```
#include "im_infra.h"

int image_ok(IMAGE *image)
```

***DESCRIPTION***

The pointer "image" is checked if it points to a valid image. The linked list in which all the images are present is scanned for the occurrence of "image". If no image exist with this pointer, an error is generated and the following message is added to the error-stack:

Non existing image pointer.

The function is_image() performs the same check and has the same return values but does not put an alert box on the screen.

***RETURN VALUES***

IE_OK (1) if the pointer is a valid image.
IE_NOT_OK (0) if the pointer is not an image.

***SEE ALSO***

images_ok  is_image

---

## *image_output*

***NAME***

image_output - show formatted output

***SYNOPSIS***

```
#include "image.h "
#include "im_infra.h"

void image_output(int stream, const char *format, ...)
```

***DESCRIPTION***

To enable the image processing routines to be used under any user-interface, all textual messages from these functions must be visualized by the user-interface, not by the image processing. Therefore printf() and other standard output functions should not be used. The image_output() function is used in the image processing functions to channel all the textual information to the user-interface. image_output() is called almost the same way as printf() except that  a parameter "stream" has been inserted to identify the type of information. At present, the following types have been defined:

| | |
|---|---|
| IMO_OUTPUT: | ordinary-output (e.g.) measurement results |
| IMO_INSTRUCT: | instruction messages |
| IMO_WARING: | warning messages |
| IMO_ERROR : | error-messages |

How these different streams are shown to the user, is entirely up to the user-interface e.g. in SCIL_Image the error stream is visualized by means of a alert_box showing the message. Execution is resumed after the alert_box has been removed by the user.

***RETURN VALUES***

im_set_output_handler

## *image_readwrite_ok*

***NAME***

image_readwrite_ok - check if an image is writeble

***SYNOPSIS***

```
#include "im_infra.h"

int image_readwrite_ok(IMAGE *image)
```

***DESCRIPTION***

The pointer "image" is checked if it does not have the READ_ONLY flag set. If the image is read-only, an error is generated and the following message is added to the error-stack:

Illegal output image:  <imagename> (read only)

***RETURN VALUES***

IE_OK (1) on success
IE_NOT_OK (0) when the image is read-only

---

## *image_text*

***NAME***

image_text - generate text within an image

***SYNOPSIS***

```
#include "im_proto.h"

int image_text(IMAGE *out, int x, int y, int val, int boxval, int
zoom, char *str)
```

***DESCRIPTION***

Write the text string specified by "str" into the image "out", starting at relative position (x,y). The intensity of the characters is specified by "val", the background intensity is "boxval". "zoom" specifies the zoom factor.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

---

## *image_to_chaincode*

***NAME***

image_to_chaincode - convert labeled skeleton into chain-code list

***SYNOPSIS***

```
#include "grey_2dp.h"

int image_to_chaincode(IMAGE *image, VAR_OBJECT *output)
```

***DESCRIPTION***

The labeled image "image" is scanned for objects, which should be 8-connected skeletons, and a Freeman chain-code description of the objects is stored into the 1-dimensional VAR_OBJECT "output" of type SHORT_T.

Freeman chain-codes

Freeman chain-codes describe an 8-connected curve in the following way. The coordinates of the first point of the curve are stored. Then for all points of the curve, the direction to the next point is stored (Freeman code). In a rectangular grid, an 8-connected curve can only step in 8 different directions, so the directions are coded as:

```
3  2  1
4  *  0
5  6  7
```

In this way, an 8-connected curve can be represented by:

the x-coordinate of the first point,
the y-coordinate of the first point,
the number of Freeman codes, N,
N Freeman codes.

If the curve is closed, N is equal to the number of pixels on the curve: the last chain-code points to the first point. If the curve is open, N equals the number of points minus one, as there is no successor to the last point.

Each object in "image" is parsed into a set of curve segments, connecting two endpoints or branch-points or being a closed loop. Each curve segment is then represented by a starting point and a set of chain-codes.

The representations of the objects are stored into "output" in the following order:

the number of objects stored,
the representation of the first object,
the representation of the second object,
...

The representation of each object is as follows:

the number of curves in the object,
the representation of the first curve,
the representation of the second curve,
...

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

chaincode_to_image  chaincode_to_xy  put_xy_into_image

---

## ***image_to_var_object***

***NAME***

image_to_var_object - convert an image into a var_object

***SYNOPSIS***

```
#include "objectsp.h"

int image_to_var_object(IMAGE *image, VAR_OBJECT *object, int
type_of_object)
```

***DESCRIPTION***

Convert the image "image" into a the var_object "object". "type_of_object" specifies the type that the object will become. If "type_of_object" is zero, then the type of the object itself will be taken.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

var_object  var_object_convert  var_object_to_image

---

## *ImageMotionEvents*

***NAME***

ImageMotionEvents - enable/disable motion events for an image

***PLATFORM***

UNIX.

***SYNOPSIS***

```
#include "disp_p.h"

int ImageMotionEvents(IMAGE *image, int mode)
```

***DESCRIPTION***

An interactive application must react to all events generated by the windowing system in order to perform its tasks. However not all events that can be generated by a windowing system are of interest to certain applications. One of these events is the ButtonMotionEvent generated by the X-windows system. The number of this type of event can be quite large, decreasing the performance of the application and creating undesired delayed feedback effects. ImageMotionEvents() enables the programmer to turn of this event for the window attached to "image". To turn off these events, "mode" must be set to No (0). The events are turned back on again by mode Yes (1).

Any application that turns off these events, should turn them back on again before the application ends.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

im_input_func  del_im_input_func  im_exposure_func  del_im_exposure_func
set_image_interaction  handle_events  point_im  poll_mouse

## *images_ok*

***NAME***

images_ok - check if the two pointers supplied are valid images

***SYNOPSIS***

```
#include "im_infra.h"

int images_ok(IMAGE *image1, IMAGE *image2)
```

***DESCRIPTION***

The pointers "image1" and "image2" are checked if they point to valid images. The linked list in which all the images are present is scanned for the occurrence of "image1" and "image2". If not both pointers are valid images, an error is generated and the following message is added to the error-stack:

Non existing image pointer.

***RETURN VALUES***

IE_OK (1) if both pointers are valid images
Negative error status on failure (see im_error.h)

***SEE ALSO***

image_ok  is_image

---

## *imaginary_im*

***NAME***

imaginary_im - get the imaginary part of an complex image

***SYNOPSIS***

```
#include "im_proto.h"

int imaginary_im(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

Take the imaginary part of each element of the image "in" (a complex image) and store the results in the image "out". If "out" is a complex image then the result will be stored in the real part of each element of "out" and the imaginary part will be cleared. If "out" is not a complex image the result will be a float image "out".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

real_im  complex_im

---

## *Imeasure*

***NAME***

Imeasure - Interactive object measurements

***SYNOPSIS***

```
#include "im_aio.h"

LIST *Imeasure(IMAGE *grey, IMAGE *binary, int garb, unsigned long
shape, unsigned long dens, int print_it, char *file)
```

***DESCRIPTION***

| | | |
|---|---|---|
| grey | - | Grey value image containing original object |
| binary | - | Binary image containing mask of the objects |
| garb | - | Object garbage level |
| print_it | - | Print results |
| shape, dens | - | Bitmaps with feature specification |
| file | - | Store results in file |

Imeasure() is the interactive measurement routine of the AIO package.

The function list_label() is used to label the objects in the binary image using 8 connectivity and a garbage level "garb". Then the functions object_shape_meas() and object_dens_meas() are used to measure the shape and densitometry features specified in the "shape" and "dens" bitmaps. The results of the measurements are shown on your terminal/worksheet if "print_it" is 1. If a filename other than "-" is given in "file" the results are stored in that file.

***RETURN VALUES***

A list with object information is returned on success
NULL on failure.

***SEE ALSO***

measure  object_shape_meas object_dens_meas list_label

## *increment_im*

***NAME***

increment_im - increment

***SYNOPSIS***

```
#include "im_proto.h"

int increment_im(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

Increment each element of "in" and store the result in the corresponding element of "out".

***NOTE***

For more powerful image arithmetic expressions (scaling, adding offsets, etc.), use the function eval() .

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

decrement_im  eval

## *init_func_overload*

***NAME***

init_func_overload - initialize the function overload tables

***SYNOPSIS***

```
#include "im_infra.h"

int init_func_overload(void)
```

***DESCRIPTION***

init_func_overload() initializes the function overload tables used by the function overload mechanism of Image. If init_func_overload() is not done the overload mechanism will not work.

***NOTE***

This routine is intended for system administrational use only.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

show_func_overload  overloadable_func

---

## *init_scil_image*

***NAME***

init_scil_image - perform necessary initialization of the SCIL_Image package

***SYNOPSIS***

```
void init_scil_image(void)
```

***DESCRIPTION***

init_scil_image() performs the necessary initialization for the correct operation of the SCIL_Image environment.. It must be present in the "scilinit" executed before any image-processing operation is performed.

***RETURN VALUES***

None

***SEE ALSO***

default_images

---

## *init_silo*

***NAME***

init_silo - initialize image-silo package

***SYNOPSIS***

```
void init_silo(void)
```

***DESCRIPTION***

Initializes the silo-package. The silo_err variable is defined here. An array of binary masks used to maintain an entry list is filled here.

***RETURN VALUES***

None

---

## *initimage*

***NAME***

initimage - initialize the image processing package

***SYNOPSIS***

```
#include "im_infra.h"

int initimage(void)
```

***DESCRIPTION***

initimage() performs necessary initialization for correct operation of the image infrastructure. It must be called before any image-processing operation is executed.

***RETURN VALUES***

None

---

## *interpret*

***NAME***

interpret - interpret a command

***SYNOPSIS***

```
void interpret(char *str)
```

***DESCRIPTION***

interpret() sends the given string "str" to the SCIL command interpreter. The string is then interpreted as if it was typed at the keyboard.

***RETURN VALUES***

None

---

## *intlow*

***NAME***

intlow

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See lowest_int

---

## *invert_im*

***NAME***

invert_im - bitwise inversion of image pixels

***SYNOPSIS***

```
#include "im_proto.h"

int invert_im(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

Perform a bitwise invert operation of each element of "in" and store the result in "out"

***NOTE***

For more powerful image arithmetic expressions (scaling, adding offsets, etc.), use the function eval().

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

and_im  or_im  xor_im  shift_im

---

## *Irectangle*

***NAME***

Irectangle - interactive box selection

***SYNOPSIS***

```
#include "itools.h"

int Irectangle(IMAGE **im, char *mess, int *x, int *y, int *z, int
*w, int *h, int *d)
```

***DESCRIPTION***

Irectangle() enables the user to select a rectangular region in an image. First the user is shown the string "mess" as a message to start dragging; if "mess" is NULL or an empty string, no message is shown. After the user has drawn a rectangle in "im", and (for 3D) extended the Z slice, the coordinates of the rectangle are returned in "x", "y" and "z", and the sizes of the rectangle in "w","h" and "d". If "im" or "*im" is NULL, every image may be pointed in; in that case the image in which the rectangle is drawn is returned in "*im". The user may interrupt pointing by pressing a key, which will be returned. The visual feedback given during selection is removed at the end of selection.

***RETURN VALUES***

The key pressed or 0

***SEE ALSO***

point_im

## *is_image*

***NAME***

is_image - tell if the supplied pointer is a image (no warning)

***SYNOPSIS***

```
#include "im_infra.h"

int is_image(IMAGE *image)
```

***DESCRIPTION***

The pointer "image" is checked against the linked list of existing images and if it is a valid image a true status is returned. If it is not an image then just a false status is returned, no error is generated. This function and the function image_ok() perform the same check and have the same return status, except that image_ok() does generate an error if the pointer is not an image.

The function is meant for situations that a pointer can point to several different structure (or to nothing at all) and the software has to find out where it is pointing at.

***RETURN VALUES***

TRUE (1) if the pointer "image" is a valid image.
FALSE (0) if not.

***SEE ALSO***

image_ok  images_ok

## *is_var_object*

***NAME***

is_var_object - tell if the pointer is a var_object (no warning)

***SYNOPSIS***

```
#include "objectsp.h"

int is_var_object(VAR_OBJECT *var_object)
```

***DESCRIPTION***

The pointer "var_object" is checked against the linked list of existing var_objects and if it is a valid var_object a true status is returned. If it is not an var_object then just a false status is returned, no warning is displayed on the screen. This function and the function var_object_ok() perform the same check and have the same return status, except that var_object_ok() does put a warning on the screen.

The function is meant for situations that a pointer can point to several different structure (or to nothing at all) and the software has to find out where it is pointing at.

***RETURN VALUES***

TRUE (1) if the pointer "var_object" is a valid var_object.
FALSE (0) if not.

***SEE ALSO***

var_object_ok  var_object

## *isalnum*

## *isalpha*

## *isascii*

## *iscntrl*

## *isdigit*

## *isgraph*

## *islower*

## *isprint*

## *ispunct*

## *isspace*

## *isupper*

## *isxdigit*

***NAME***

isalnum, isalpha, isascii, iscntrl, isdigit, isgraph, islower, isprint, ispunct, isspace, isupper, isxdigit - character classification

***SYNOPSIS***

```
#include <ctype.h>

int isalnum(int c)

int isalpha(int c)

int isascii(int c)

int iscntrl(int c)

int isdigit(int c)

int isgraph(int c)

int islower(int c)

int isprint(int c)

int ispunct(int c)

int isspace(int c)

int isupper(int c)

int isxdigit(int c)
```

***DESCRIPTION***

These functions are interfaces function to the standard C macros as implemented on thecurrent system. The functionality of these macros is:

These macros classify ASCII-coded integer values by table lookup. Each is a predicate returning nonzero for true, zero for false. Isascii is defined on all integer values; the rest are defined only where isascii is true and on the single non-ASCII value EOF (see stdio.h).

| | |
|---|---|
| isalnum | c is an alphanumeric character |
| isalpha | c is a letter |
| isascii | c is an ASCII character, code less than 0200 |
| iscntrl | c is a delete character (0177) or ordinary control character (less than 040). |
| isdigit | c is a digit |
| isgraph | c is a printing character except space |
| islower | c is a lower case letter |
| isprint | c is a printing character, code 040 (space) through 0176 (tilde) |
| ispunct | c is a punctuation character (neither control nor alphanumeric) |
| isspace | c is a space, tab, carriage return, newline, or formfeed |
| isupper | c is an upper case letter |
| isxdigit | c is a hexadecimal digit |

***RETURN VALUES***

The functions all return a non-zero (true) value if the argument c satisfies the condition described, and zero if not

## *IsMouseDown*

***NAME***

IsMouseDown - test whether a specified mouse button is down

***SYNOPSIS***

```
#include "disp_p.h"

int IsMouseDown(IM_EVENT mouse_event, int button)
```

***DESCRIPTION***

IsMouseDown() can be used to find out whether the specified button or buttons are down. The "button" argument can be a combination of the symbolic values LEFT, MIDDLE, RIGHT, separated by "|" (the bitwise OR). The function returns the "bitwise OR" of the buttons which were specified and being down (e.g. if the button argument reads "LEFT | MIDDLE | RIGHT" and the buttons being down are LEFT and RIGHT, the value "LEFT | RIGHT" will be returned. IsMouseDown() can only be used after a call to the "point_im" routine which returns a mouse-event as one of its arguments.

***EXAMPLE***

```
#include "disp_p.h"
#include "image.h"

IMAGE    *ip;
int       x, y;
int       val;
IM_EVENT  event;

printf("Left Middle Right\n");
while (point_im(&ip, &x, &y, &event) != ’q’) {
      val = IsMouseDown(event, LEFT | RIGHT | MIDDLE);
      printf(" %d %d %d\r", (val&LEFT)>0, (val&MIDDLE)>0,
(val&RIGHT)>0);
      fflush(stdout);
}
```

***RETURN VALUES***

"Bitwise OR" combination of LEFT, MIDDLE and RIGHT buttons being down.
0 if none of the specified button(s) was down.

***SEE ALSO***

point_im  MousePress  MouseRelease  MouseMove  EventType  KeyPressed

## *isodata_threshold*

***NAME***

isodata_threshold - thresholding using the isodata algorithm.

***SYNOPSIS***

```
#include "im_proto.h"

int isodata_threshold(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

Perform thresholding operation on the grey value image "in" and store the result in the binary image "out". The threshold-level is determined by the isodata algorithm. This algorithm is an iterative method based upon the grey level histogram of the image. The histogram is split up into two parts, the foreground pixels and the background pixels, assuming an initial threshold value. Then the average value of the foreground and of the background pixels is calculated and a new threshold value is taken midway between those two values.
This process is repeated, based upon the new threshold estimate until the threshold value does not change any more.

***RETURN VALUES***

The used threshold value.

***SEE ALSO***

threshold  entropy_threshold

## ***iter_ok***

***NAME***

iter_ok - check if the number of iterations is positive or zero

***SYNOPSIS***

```
#include "im_infra.h"

int iter_ok(int iter)
```

***DESCRIPTION***

The function checks if the number "iter" is zero or greater than zero. If it is not an error is generated and the following message is added to the error-stack:

```
Nr of iterations [<iter>] must be a positive number
```

The function is used in operations that have an argument that specifies a number of iterations.

***RETURN VALUES***

IE_OK (1) if the number is positive or zero
IE_NOT_OK (0) if the number is negative

***SEE ALSO***

range_ok  positive_ok

## *IThreshold*

***NAME***

IThreshold - interactive threshold editor

***SYNOPSIS***

```
#include "itools.h"

void IThreshold(IMAGE *image)
```

***DESCRIPTION***

IThreshold() is an interactive threshold editor with two threshold levels. The histogram of the image "image" is shown in a separate window. The two vertical lines that run through the histogram of the image represent the threshold levels, the numbers printed above them are the number of pixels equal to those levels. These two lines are part of the scrolling mechanism present under the histogram which can be used to change the threshold levels. The result of thresholding with the two levels can be seen simultaneously in the image through the use of the colors green, red and blue. The pixels that have a value equal to the lower level are shown in red, the pixels equal to the upper level in blue and the pixels in between in green. Pixels outside both levels keep their original appearance.

To move the scrolling mechanism, click the mouse button and drag the middle part. Both the threshold levels will move synchronously. Dragging with the mouse button in the left part of the mechanism will alter the lower level only. The upper level can be moved by dragging the right part of the scrolling mechanism.

At the bottom of the threshold editor three buttons are located with the following functions:

| | |
|---|---|
| "OK" | Exit the editor, threshold the image according to the two levels set. |
| "CANCEL" | Exit the editor, leave the image intact. |
| "HELP" | Short on-line help. |

***RETURN VALUES***

None

***SEE ALSO***

IGreyMap

## *jpeg_readfile*

## *jpeg_writefile*

## *set_jpeg_quality*

### *NAME*

jpeg_readfile - read an image from a file in JPEG (JFIF) format

jpeg_writefile - write an image to a file in JPEG (JFIF) format

set_jpeg_quality - change the JPEG quality percentage

### *SYNOPSIS*

```
#include "im_proto.h"

IMAGE *jpeg_readfile(char *filename, IMAGE *image, int xpos, int
ypos)

int jpeg_writefile(IMAGE *image, char *filename)

int set_jpeg_quality(int percentage)
```

### *DESCRIPTION*

jpeg_readfile() reads the image stored in the JPEG file "filename" and puts it in image "image". If "USE_NAME" (a NULL pointer) is specified as the image, a new image is created at position "xpos", "ypos", with the same name as the file. If an image is already present with that name, that image will be re-used.

jpeg_writefile() writes the image "image" to the file "filename" using the JPEG format.

These functions are capable of handling JPEG-files according to the JPEG File Interchange Format (JFIF) specifications.

The obligatory filename extension for this file format is ".jpg".

By default the image quality is set at 75%. To change this percentage for subsequent writes, use the set_jpeg_quality() function.

### *RETURN VALUES*

jpeg_readfile() returns the pointer to the image in which the data was put, either an existing image or a newly created one. It returns NULL on failure to load the file.

jpeg_writefile() and set_jpeg_quality() :

IE_OK (1) on success
Negative error status on failure (see im_error.h)

### *SEE ALSO*

writefile  readfile

## *karhunen_loeve*

## *im_eigenvectors*

## *im_principle_component*

***NAME***

karhunen_loeve, im_eigenvectors, im_principle_component - principle component analysis of image planes

***SYNOPSIS***

```
#include "im_proto.h"

int karhunen_loeve(IMAGE *in, IMAGE *out, int start, int end)

int im_eigenvectors(IMAGE *in, VAR_OBJECT *vecs, VAR_OBJECT *vals)

int im_principle_component(IMAGE *in, VAR_OBJECT *vecs, IMAGE *out,
int nr)
```

***DESCRIPTION***

These functions apply an eigenvector analysis on the planes of image "in". This 3D input image is interpreted as a set of 2D image "features". The karhunen_loeve() function returns directly the "start" to "end" principle components of the input, where the first principle component has the largest eigenvalue and the last (=ImageDepth(in)-1) principle component the smallest. The im_eigenvectors() function returns the eigenvectors "vecs" and eigenvalues "vals" in the corresponding var_objects (DOUBLE_T). After analysis, the components can be extracted by applying im_principle_component(). The "vecs" var_object is the one returned from im_eigenvectors(); "nr" determines which component will be returned in "out".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

covplanematrix

## *KeyPressed*

***NAME***

KeyPressed - test whether a key has been pressed inside an image

***SYNOPSIS***

```
#include "disp_p.h"

int KeyPressed(IM_EVENT event)
```

***DESCRIPTION***

KeyPressed() can be used to find out whether a key has been pressed inside an image. It returns the ASCII value of the key pressed or 0 (zero) if no key was pressed. KeyPressed can only be used after a call to the "point_im" routine which returns an event as one of its arguments.

***EXAMPLE***

```
#include "disp_p.h"
#include "image.h"

IMAGE    *ip;
int       x, y;
int       val;
IM_EVENT  event;

while (point_im(&ip, &x, &y, &event) != ’q’) {
      val = KeyPressed(event);
      if(val) printf("[%c]", val);
}
```

***NOTE***

The KeyPressed mechanism ensures the mapping of carriage return to new-line, no matter what mode the terminal is in.

***RETURN VALUES***

The ASCII value of the key pressed.
0 if no key pressed.

***SEE ALSO***

point_im  MousePress  MouseRelease  MouseMove  IsMouseDown  EventType

## *kirsch_temp*

***NAME***

kirsch_temp - edge detection filter

***SYNOPSIS***

```
#include "im_proto.h"

int kirsch_temp(IMAGE *in, IMAGE *out, IMAGE *direction, int flag)
```

***DESCRIPTION***

Template type edge detection based upon the Kirsch operator. Within the moving window in the image "in", with dimensions 3 * 3, eight convolutions with the following masks are calculated:

$$\begin{array}{c}(0)\\ \begin{matrix}-3 & -3 & 5\\ -3 & 0 & 5\\ -3 & -3 & 5\end{matrix}\end{array}\quad\begin{array}{c}(1)\\ \begin{matrix}-3 & 5 & 5\\ -3 & 0 & 5\\ -3 & -3 & -3\end{matrix}\end{array}\quad\begin{array}{c}(2)\\ \begin{matrix}5 & 5 & 5\\ -3 & 0 & -3\\ -3 & -3 & -3\end{matrix}\end{array}\quad\begin{array}{c}(3)\\ \begin{matrix}5 & 5 & -3\\ 5 & 0 & -3\\ -3 & -3 & -3\end{matrix}\end{array}$$

$$\begin{array}{c}(4)\\ \begin{matrix}5 & -3 & -3\\ 5 & 0 & -3\\ 5 & -3 & -3\end{matrix}\end{array}\quad\begin{array}{c}(5)\\ \begin{matrix}-3 & -3 & -3\\ 5 & 0 & -3\\ 5 & 5 & -3\end{matrix}\end{array}\quad\begin{array}{c}(6)\\ \begin{matrix}-3 & -3 & -3\\ -3 & 0 & -3\\ 5 & 5 & 5\end{matrix}\end{array}\quad\begin{array}{c}(7)\\ \begin{matrix}-3 & -3 & -3\\ -3 & 0 & 5\\ -3 & 5 & 5\end{matrix}\end{array}$$

The output value is the maximum of the results of all these convolutions. It is stored into "out", in the pixel corresponding with the central pixel of the window. The sequence number of the convolution mask with the maximum result is an estimate of the direction of the first derivative and it is stored in the image "direction", if this is specified (flag = 1).

***RETURN VALUES***

IE_OK (1) on success  
Negative error status on failure (see im_error.h)

***SEE ALSO***

laplace prewitt_temp robinson_temp prewitt_diff roberts_diff sobel_diff

## *kuwahara*

***NAME***

kuwahara - edge preserving smoothing (Kuwahara)

***SYNOPSIS***

```
#include "im_proto.h"

int kuwahara(IMAGE *in, IMAGE *out, int fsize)
```

***DESCRIPTION***

Perform an edge preserving smoothing (Kuwahara filter) on the pixels of image "in" and store the result in image "out".
Image "in" is scanned with a moving window with dimensions "fsize" * "fsize".
The command subdivides the moving window into four sub-windows. In each sub-window the variance of the pixel values is calculated. The window with the lowest variance is taken as the averaging window. The resulting average value is stored in the pixel in image "out" that corresponds with the central pixel in the moving window.
This approach tends to avoid the sub-windows with large variations in pixel values, e.g. due to the occurrence of an edge.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

edge_preserve

## *kuwahara_round*

***NAME***

kuwahara_round - kuwahara filter using a circular filter window

***SYNOPSIS***

```
#include "im_proto.h"

int kuwahara_round(IMAGE *in, IMAGE *out, int fsize)
```

***DESCRIPTION***

Perform an edge preserving smoothing (Kuwahara filter) on the pixels of image "in" and store the result in image "out". Image "in" is scanned with a moving circular window with diameter "fsize".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

grey_morph_round  kuwahara  vkuwahara

## ***label***

***NAME***

label - image labeling

***SYNOPSIS***

```
#include "im_proto.h"

int label(IMAGE *in, IMAGE *out, int conn)
```

***DESCRIPTION***

Subdivide binary image "in" into different components, based upon connectivity analysis, label each object with a sequence number, assign this sequence number as a new pixel value to each pixel of the object and store the resulting image in the labeled image "out". "conn" is the connectivity which can be either 4 or 8..

For 2D images the maximum number of objects that can be present in the image is limited by the maximum label number that can be stored in the output image (short = 32767).

For 3D images the number of object that can be present in the image is limited to 4095 due to the implementation of the algorithm.

***RETURN VALUES***

The number of objects in the input image "in".

***SEE ALSO***

hull  objectsize  rhull  small_object_removal

## *laplace*

***NAME***

laplace - Laplace edge detector

***SYNOPSIS***

```
#include "im_proto.h"

int laplace(IMAGE *in, IMAGE *out, int mask)
```

***DESCRIPTION***

Differential edge detection based upon the Laplacian operator. Within the moving window in the image "in", with dimensions 3 * 3 a convolution with one of the following masks is calculated. The mask to be applied may be specified by the parameter "mask", where "mask" is 1, 2 or 3 corresponding with the following masks:

```
     (1)            (2)            (3)
 0  -1   0      -1  -1  -1       1  -2   1
-1   4  -1      -1   8  -1      -2   4  -2
 0  -1   0      -1  -1  -1       1  -2   1
```

The convolution result of the selected mask is stored into the corresponding central pixel of the image "out".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

prewitt_temp  kirsch_temp  robinson_temp  prewitt_diff  roberts_diff  sobel_diff

## *laxis*

***NAME***

laxis - obtain long axis of fitted ellipse of object

***SYNOPSIS***

```
#include "im_aio.h"

double laxis(LIST *link)
```

***DESCRIPTION***

| | | |
|---|---|---|
| link | - | Link pointing to object |

AIO primitive to obtain value of an object feature

laxis() returns the length of the long axis of fitted ellipse of the object pointed to by "link" if this has previously been measured.

***RETURN VALUES***

length of the long axis of fitted ellipse of object on success
0.0 if link is not an object or if long axis has not been measured

***SEE ALSO***

measure  object_shape_meas  object_dens_meas  saxis

## lens

## stop_lens

***NAME***

lens, stop_lens - interactive part image processing

***SYNOPSIS***

```
#include "im2scil.h"

int lens(IMAGE *output, int width, int height, char *command)

int stop_lens(void)
```

***DESCRIPTION***

lens() is a function, which allows interactive image processing on small part images of dimensions "width" * "height".

In lens() when the user to points to any of the visible images a small rectangle of dimensions "width" * "height" is copied to the output image, and the specified command string is executed. If command is 0 only the copy action is performed.

The lens() command is especially useful when trying out new algorithms which are still in interpreted rather than in compiled form. Since lens() lets you choose interesting parts of the image the data to process can be kept small, and the slower interpreted speed can be used without to much frustration. It certainly makes a difference in SCIL whether you are operating on an 31*31 image, or on a 256*256 image.

To stop lens() either give a <RETURN> at the keyboard or use the function stop_lens().

stop_lens() halts lens(), which normally keeps on running until a <RETURN> is typed at the keyboard.

***EXAMPLE***

To use lens() as a simple magnifier, only the copy action of lens() is needed, since the user can resize the output image at will. The following can be typed:

```
readf bnoise.im
lens(B, 41, 41, NULL);
```

Now pointing at image A will result in a magnification of the pointed area displayed in image B.

The following example illustrates the use of lens in combination with a command. In the example noise removal through percentile filtering can be done almost in real time:

```
int small;
small = make_image small GREY_2D 64 64;
lens(small, 31, 31, "percentile small small");
```

***RETURN VALUES***

None.

***SEE ALSO***

copy_part_image

---

## *life*

***NAME***

life - Conway’s game of life

***SYNOPSIS***

```
#include "im_proto.h"

int life(IMAGE *in, IMAGE *out, int iter, int bound)
```

***DESCRIPTION***

Calculate the next generation(s) from a given generation from image "in" according to the rules of the "Conway’s game of life". The result is stored in the image "out". "iter" can be used to calculate several generations in one cycle. "bound" specifies that the edge around the image must be set to foreground (1) or to background (0) pixels.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

---

## *list*

## *more*

***NAME***

list, more - show contents of the program buffer

***SYNOPSIS***

```
list [start],[end]

more [start],[end]
```

***DESCRIPTION***

The loaded text program can be inspected with the similar commands "list" or "more". The difference between "more" and "list" is that the first offers the text interactively in chunks while the latter just bulks away. "more" asks what to do at the end of each page. The following answers are possible:

| | |
|---|---|
| <SPACE> | hitting the space bar will show the next full page on the screen |
| <CR> | shows one more line |
| d | shows half a page more |
| q | prevents more from more |

***EXAMPLE***

```
 [C1] list        list all
 [C2] list  5     displays line 5 of program text
 [C3] list  5,10  lists lines from 5 to 10
 [C4] list   ,10  lists from start to line 10
 [C5] list 10,    lists from line 10 until end
```

Instead of separating "start" and "end" with a comma a space is also allowed.

## *list_cluts*

***NAME***

list_cluts - display a list of all cluts

***SYNOPSIS***

```
#include "im_infra.h"

int list_cluts(void)
```

***DESCRIPTION***

"list_cluts" generates a list terminal of all cluts that are available on the controlling.

***RETURN VALUES***

IE_OK (1)

***SEE ALSO***

create_clut  clut_by_name

## list_label

***NAME***

list_label - label image and create list with objects

***SYNOPSIS***

```
#include "im_aio.h"

LIST *list_label(IMAGE *in, IMAGE *out, int con, int garb)
```

***DESCRIPTION***

| | | |
|---|---|---|
| in | - | Pointer to binary image |
| out | - | Pointer to output image |
| con | - | Connectivity (4/8) |
| garb | - | Garbage level |

list_label() labels the objects in the input image "in". Objects smaller than "garb" pixels are not labeled and do not appear in the output image "out". "con" is the connetivity used for the labeling, values can be 4 or 8. During the labeling a linked list is created with all the objects. This list can later be used to manipulate or measure the objects according to the AIO concept.

The recursive labeling algorithm tries not to use the same label on a horizontal scanline.

During the labeling process some object information is automatically obtained and placed in the object list:

| | | |
|---|---|---|
| xmin, xmax | - | Minimum and maximum x coordinate |
| ymin, ymax | - | Minimum and maximum y coordinate |
| area | - | Number of object pixels |

***NOTE***

list_label is part of the AIO package. It is the users responsibility to remove the list with "rm_list(list)"

***EXAMPLE***

```
To label object bigger than 20 pixels, measure the perimeter
and hide those objects whose perimeter is bigger than 70.0:

#include "image.h "
#include "im_aio.h"
LIST *l, *o;

readfile("cermet",a,0,0);
threshold(a,b,128);
invert_im(b,b);
l = list_label(b,c,8,20);
FORALL(o,l) object_shape_meas(c,o,PERI);
FORALL(o,l) if(peri(o) > 70.0) hide_object(c,o);
display_image(c);
l = rm_list(l);
```

***RETURN VALUES***

A list with information on the labeled objects is returned.

***SEE ALSO***

aio_label  rm_list

---

## *list_var_objects*

***NAME***

list_var_objects - display a list of all var_objects

***SYNOPSIS***

```
#include "objectsp.h"

int list_var_objects(void)
```

***DESCRIPTION***

"list_var_objects" displays a list of all var_objects and their classes on the controlling terminal.

***RETURN VALUES***

Always IE_OK (1)

***SEE ALSO***

var_object  destroy_var_object  show_var_object_info  var_object_by_name

---

## *lmax*

***NAME***

lmax

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See grey_dilation

---

## *lmin*

***NAME***

lmin

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See grey_erosion

---

## *ln_im*

***NAME***

ln_im - natural logarithm

***SYNOPSIS***

```
#include "im_proto.h"

int ln_im(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

Calculate the natural logarithm of each element of "in" and store the result in the corresponding element of "out".

***NOTE***

For more powerful image arithmetic expressions (scaling, adding offsets, etc.), use the function eval().

***RETURN VALUES***

Number of domain conflicts (number of negative or zero pixels in the input image), so 0 is OK.
Negative error status on failure (see im_error.h).

***SEE ALSO***

exp_im  log10_im

---

## *load*

***NAME***

load - load a C source code file into the interpreter

***SYNOPSIS***

```
load [filename]
```

***DESCRIPTION***

With the command "load" program text is loaded into SCIL. When no filename is given, SCIL will prompt for one. When loading a specific file the contents of a previous loaded file is overwritten, also global variables are deleted.

***EXAMPLE***

```
[C1] load demo.c

or

[C1] load
filename: demo.c
[C2]
```

Evidently both examples will load the file "demo.c".

## local_contrast

### NAME

local_contrast - contrast enhancement filter

### SYNOPSIS

```
#include "im_proto.h"

int local_contrast(IMAGE *in, IMAGE *out, int radius, int
frightmarexp, int fleftmarexp, int fleftmargin, int frightmargin)
```

### DESCRIPTION

local_contrast() is a filter that locally enhances the contrast of image "in" by evaluating the histogram in a circular window around a pixel of interest and stores the result in the image "out". The variable "radius" determines the radius of the window.

This procedure operates on the basis of a simplified from of histogram equalization: assuming the histogram is sparse (number of pixels in window is low compared to grey-value range). See below for details about processing of large dynamic range images. Equalization is then realized by numbering the non-empty bins starting from zero. The numbered bins can be thought to present an equalized histogram. Due to the sparseness and image noise, the number of bins will, usually, not vary much.

The resulting histogram is stretched using a pre-determined constant (see below). Its center (median) is shifted to coincide with the median of the original histogram. Subsequently, the value of the "central" pixel is traced to its location in the equalized histogram and form there to the stretched histogram.

To understand why this results in contrast enhancement, consider two neighboring pixels: Since the windows around these pixels largely overlap, the range of both histograms will be roughly the same. As a result, the overall expansion factor from original histogram to stretched equalized the two pixel-values will be multiplied with the expansion factor: contrast is enhanced. In image areas where the range original histogram was large (edges) no stretching occurs; contrast may even be reduced depending on the value of the stretching factor. The positions of the margins is controlled through "leftmargin" and "rightmargin"; the stretch values at these positions are controlled through "frightmarexp" and "fleftmarexp".

A special condition occurs when shifting of the histogram would result in (stretched) histogram bins outside the range in the original image. In these cases shifted is limited to avoid this situation. In some applications it is desirable to enhance contrast only in light or dark areas of the image. This is achieved by making the stretch factor dependent on the median in the original histogram. This dependency has the form of a simple linear mapping between two bound or "margins". Outside the margins the stretch factor is equal to the value at the nearest margin.

Processing of large dynamic range images.

Images with a grey-value range larger than 0..255 are assumed to be recorded using photon counting or equivalent sensors. Since the procedure will not work well on images where the variance is strongly dependent on the signal level (as in photon limited images), the input image is transformed by taking the square root of the pixels values. Subsequently, the operation is performed and the transform is reversed.

***LITERATURE***

van der Voort, H.T.M, G.J. Brakenhoff, J.A.C. Valkenburg & N.Nanninga. 1985. Design and use of a computer- controlled confocal microscope. Scanning 7: 66-78.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

homomorphic

## *local_glc_entropy*

## *local_glc_contrast*

## *local_glc_asymmetry*

***NAME***

local_glc_entropy - local texture measure, co-occurence of greylevels
local_glc_contrast - local texture measure, co-occurence of greylevels
local_glc_asymmetry - local texture measure, co-occurence of greylevels

***SYNOPSIS***

```
#include "im_proto.h"

int local_glc_entropy(IMAGE *input, IMAGE *output, int fwidth, int
fheight int vectorx, int vectory)

int local_glc_contrast(IMAGE *input, IMAGE *output, int fwidth, int
fheight, int vectorx, int vectory)

int local_glc_asymmetry(IMAGE *input, IMAGE *output, int fwidth, int
fheight, int vectorx, int vectory)
```

***DESCRIPTION***

The functions calculate a local 2-dimensional histogram of the combinations of greyvalues of pixels that are the startpoint/endpoint of a vector with a specified ("vectorx","vectory") displacement. Calculation takes place in a rectangle "fwidth"*"fheight" around each centre pixel. The calculated value is stored in image "output".

The functions calculate:

| | |
|---|---|
| local_glc_asymmetry | Asymmetry of the histogram<br>Sum over g1,g2 of (p(g1,g2)**2) |
| local_glc_contrast | Contrast of the histogram<br>Sum over g1,g2 of ((g1-g2)**2)*p(g1,g2) |
| local_glc_entropy | Entropy of the histogram<br>Sum over g1,g2 of p(g1,g2)*log(p(g1,g2)) |

Where g1 and g2 are the grey-values at the start and end of the vector, and p(g1,g2) is the chance of this combination in this local "fwidth"*"fheight" part of the image.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

local_gld_mean local_gld_entropy local_gld_contrast local_gld_asymmetry
local_glr_nonuniformity local_glr_shortrunemphasis
local_glr_longrunemphasis local_glr_greynonuniformity local_glr_percentage

glc_asymmetry  glc_contrast  glc_entropy

## *local_gld_mean*

## *local_gld_entropy*

## *local_gld_contrast*

## *local_gld_asymmetry*

***NAME***

local_gld_mean - local texture measure, difference of greylevels
local_gld_entropy - local texture measure, difference of greylevels
local_gld_contrast - local texture measure, difference of greylevels
local_gld_asymmetry - local texture measure, difference of greylevels

***SYNOPSIS***

```
#include "im_proto.h"

int local_gld_mean(IMAGE *input, IMAGE *output, int fwidth, int
fheight, int vectorx, int vectory)

int local_gld_entropy(IMAGE *input, IMAGE *output, int fwidth, int
fheight, int vectorx, int vectory)

int local_gld_contrast(IMAGE *input, IMAGE *output, int fwidth, int
fheight, int vectorx, int vectory)

int local_gld_asymmetry(IMAGE *input, IMAGE *output, int fwidth, int
fheight,int vectorx, int vectory)
```

***DESCRIPTION***

The functions calculate a local histogram of the absolute differences of all combinations of pixels that are the startpoint/endpoint of a vector with a specified ("vectorx","vectory") displacement. Calculation takes place in a rectangle "fwidth"*'fheight" around each centre pixel. The calculated value in the image" output".

The functions calculate:

local_gld_mean Mean of the histogram
Sum of i*p(i)

local_gld_entropy Entropy of the histogram
Sum of p(i)*log(p(i))

local_gld_contrast Contrast of the histogram
Sum of (i**2)*p(i)

local_gld_asymmetry Asymmetry of the histogram
Sum of (p(i)**2)

Where i is the absolute difference, and p(i) is the chance of that absolute difference in the local "fwidth"*'fheight" part of the image.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

local_glc_entropy  local_glc_contrast  local_glc_asymmetry
local_glr_nonuniformity  local_glr_shortrunemphasis
local_glr_longrunemphasis  local_glr_greynonuniformity  local_glr_percentage
gld_mean  gld_entropy  gld_contrast  gld_asymmetry

## *local_glr_nonuniformity*

## *local_glr_shortrunemphasis*

## *local_glr_longrunemphasis*

## *local_glr_greynonuniformity*

## *local_glr_percentage*

### *NAME*

local_glr_nonuniformity - local texture measure, runlength statistics
local_glr_shortrunemphasis - local texture measure, runlength statistics
local_glr_longrunemphasis - local texture measure, runlength statistics
local_glr_greynonuniformity - local texture measure, runlength statistics
local_glr_percentage - local texture measure, runlength statistics

### *SYNOPSIS*

```
#include "im_proto.h"

double local_glr_nonuniformity(IMAGE *input, IMAGE *output, int
fwidth, int fheight)

double local_glr_shortrunemphasis(IMAGE *input, IMAGE *output, int
fwidth, int fheight)

double local_glr_longrunemphasis(IMAGE *input, IMAGE *output, int
fwidth, int fheight)

double local_glr_greynonuniformity(IMAGE *input, IMAGE *output, int
fwidth, int fheight)

double local_glr_percentage(IMAGE *input, IMAGE *output, int fwidth,
int fheight)
```

### *DESCRIPTION*

The functions calculate a local histogram of the greyvalue/runlength combinations in the image. Calculation takes place in a rectangle "fwidth"*"fheight" around each center pixel. The calculated value is stored in the image "output".

The functions calculate:

| | |
|---|---|
| local_glr_shortrunemphasis | Runlength short run emphasis<br>Sum over i,j of p(i,j)/(j**2) |
| local_glr_longrunemphasis | Runlength long run emphasis<br>Sum over i,j of p(i,j)*(j**2) |
| local_glr_greynonuniformity | Runlength greylevel nonuniformity<br>Sqrt (Sum over i of ((Sum over j of p(i,j))**2)) |
| local_glr_nonuniformity | Runlength nonuniformity<br>Sqrt (Sum over j of ((Sum over i of p(i,j))**2)) |

| | |
|---|---|
| local_glr_percentage | Runlength percentage<br>100*(Number of runs/number of pixels) |

Where i is the greylevel, j is the runlength and p(i) is the chance of that combination in the local "fwidth"*"fheight" part of the image.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

local_gld_mean  local_gld_entropy  local_gld_contrast  local_gld_asymmetry
local_glc_entropy  local_glc_contrast  local_glc_asymmetry
glr_nonuniformity  glr_shortrunemphasis  glr_longrunemphasis
glr_greynonuniformity  glr_percentage

---

## *log10_im*

***NAME***

log10_im - 10 based logarithm

***SYNOPSIS***

```
#include "im_proto.h"

int log10_im(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

Calculate the base 10 logarithm of each element of "in" and store the result in the corresponding element of "out".

***NOTE***

For more powerful image arithmetic expressions (scaling, adding offsets, etc.), use the function eval().

***RETURN VALUES***

Number of domain conflicts (number of negative or zero pixels in the input image), so 0 is OK.
Negative error status (see im_error.h).

***SEE ALSO***

exp10_im  ln_im

---

## *logon*

## *logoff*

***NAME***

logon - open a logbook file

logoff - close a logbook file

***SYNOPSIS***

```
logon <logfile>

logoff
```

***DESCRIPTION***

The command "logon" followed by a filename "logfile" creates a logbook, to keep track of all direct commands given during the session. At a later stage this file can be used as a macro to feed the interpreter. The command logoff disconnects and closes the logbook file.

***EXAMPLE***

```
 [C1] logon last_session
```

Opens the file "last_session" and from then on echoes all direct commands into the file.

If a logon command is given while there already was a logfile connected, this file is closed first.

## lookup

***NAME***

lookup - table look-up based grey level modification

***SYNOPSIS***

```
#include "im_proto.h"

int lookup(IMAGE *in, IMAGE *out, VAR_OBJECT *table, int clip)
```

***DESCRIPTION***

Substitute image "in" pixel by pixel through table lookup and store the result in "out". The original pixel value in "in" is used as an index in the look-up table stored in the var_object "table" which must be 1 dimensional and of type PIXEL_T or SHORT_T. The value of the corresponding position in "table" is used as the new pixel value for "out". "clip" is used to avoid error conditions. If "clip" is set, no error is generated but the input value is clipped between 0 and the lenght of the "table" before it is used

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

clip  threshold  contrast_stretch  equalize  tri_state_threshold

## *lower_gskeleton*

## *upper_gskeleton*

***NAME***

lower_gskeleton, upper_gskeleton - grey value skeleton

***SYNOPSIS***

```
#include "im_proto.h"

int lower_gskeleton(IMAGE *in, IMAGE *g_out, int border, int
endpixel)

int upper_gskeleton(IMAGE *in, IMAGE *g_out, IMAGE *b_out, int
metric, int border, int endpixel)
```

***DESCRIPTION***

There exists two definitions for grey-value skeletons: the upper skeleton and the lower skeleton. Pruning the branches of the upper skeleton yields a watershed. Whereas the upper skeleton always runs across the surface of the grey-value landscape, the lower skeleton may run inside. Both skeletons support options for the preservation of endpixels and setting the border of the output image ("endpixel" and "border", 0 = off, 1 = on). The upper_g_skeleton() also stores the binary skeleton image in "b_out".

The skeletons erode each elevation (grey-level) in order of increasing distance to the background. The lower_gskeleton() uses the Hilditch skeleton which is based on a city-block metric. The upper_gskeleton() uses a 3x3 Chamfer metric which allows the user to choose either the city-block metric or a pseudo Euclidian metric ("metric" = 1 = use pseudo Euclidian).

The upper skeleton is computational advantageous and is done independent of the number of grey levels (number of bits). The execution time of the lower skeleton is linear in the number of grey levels. Each level takes about the same execution time as the upper skeleton. The upper skeleton only accepts images with a maximum grey-value of 255.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

## *lowest_int*

***NAME***

lowest_int - truncate pixel values to lowest integer value

***SYNOPSIS***

```
#include "im_proto.h"

int lowest_int(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

Convert each element of image "in" into an integer value by taking the integer value just less than or equal to the original value and store the results into the corresponding elements of image "out".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

fraction_im  truncate_im  nearest_int

## *lseek*

## *tell*

***NAME***

lseek, tell - move read/write pointer

***SYNOPSIS***

```
long lseek(int fildes, long offset, int whence)

long tell(int fildes)
```

***DESCRIPTION***

These functions are interface functions to the standard C functions as implemented on the current operating system. The functionality of these functions is:

The file descriptor refers to a file open for reading or writing. The read (resp. write) pointer for the file is set as follows:

If "whence" is 0, the pointer is set to offset bytes.

If "whence" is 1, the pointer is set to its current location plus offset.

If "whence" is 2, the pointer is set to the size of the file plus offset.

The returned value is the resulting pointer location.

The function tell(fildes) is identical to lseek(fildes, 0L, 1).

***RETURN VALUES***

-1 is returned for an undefined file descriptor, seek on a pipe, or seek to a position before the beginning of file.

***SEE ALSO***

open  creat  fseek

## *macro*

***NAME***

macro - execute a macro file

***SYNOPSIS***

```
macro [-i] [-v] <macrofile>
```

***DESCRIPTION***

The command macro executes the lines in a macrofile as if they were directly typed. Therefore the macrofiles are restricted to the direct command mode.

Options:

| | |
|---|---|
| -i | every line can be executed interactively i.e. for each line the user can choose to execute it, to skip it or to quit the macro all together. |
| -v | every line is shown on the screen |

***EXAMPLE***

```
 [C1] macro -i cleanup.mac
<whatever_in_file> [y/n/q]q
 [C2]
```

## ***majority***

***NAME***

majority - majority voting

***SYNOPSIS***

```
#include "im_proto.h"

int majority(IMAGE *in, IMAGE *out, int bound, int weight)
```

***DESCRIPTION***

Performs "weighted" majority voting on image "in" and stores the result in image "out". The image is scanned by a moving 3*3 window. If the majority of the pixels within the window are object pixels, the central pixel within the window becomes an object pixel (value 1). If the majority of the pixels within the window are background pixels, the central pixel within the window becomes a background pixel (0). The "weight" parameter can be used to influence the "voting-weight" of the central pixel as shown below. Legal values for the "weight" parameter are 0, 1, 2 and 3.

```
1        1        1
1   2*weight+1    1
1        1        1
```

The threshold value used to determine the outcome of the voting is also dependent on "weight" (threshold = "weight + 5"). In case "weight" = 0, the following filter kernel is created:

```
1        1        1
1        1        1
1        1        1
```

and the threshold value becomes 5, effectively making it a median filter. Setting "weight" to 3 creates the kernel:

```
1        1        1
1        7        1
1        1        1
```

and threshold is set to 8, thus making it a true "pepper and salt removal" filter, only isolated foreground pixels are converted to background pixels and vice versa.

Choosing the values 1 or 2 for "weight. creates filters whose action on the image is somewhere in-between a median filter and a pepper-and-salt filter

"bound" specifies that the edge around the image must be set to foreground (1) or to background (0) pixels.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

## *make_color_im*

## *split_color_im*

***NAME***

make_color_im - convert three images into a color image

split_color_im - convert a color image into three separate images

***SYNOPSIS***

```
#include "color_2dp.h"

int make_color_im(IMAGE *im1, IMAGE *im2, IMAGE *im3, IMAGE *out)

int split_color_im(IMAGE *in, IMAGE *out1, IMAGE *out2, IMAGE *out3)
```

***DESCRIPTION***

make_color_im() takes the three images "im1", "im2" and "im3" and store the pixels of each image in the corresponding pixels of "out". "im1" will be stored in the red component of "out", "im2" in the green component and "im3" in the blue component of "out".

split_color_im() takes a color-image "in" and stores the red, green and blue components of each pixel into the images "out1", "out2" and "out3" respectively.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

set_color  convert

## *make_gabor*

## *standard_gabor*

## *gabor_bank*

***NAME***

make_gabor, standard_gabor, gabor_bank - make gabor filters

***SYNOPSIS***

```
#include "im_proto.h"

int make_gabor(IMAGE *out, double fcentral, double sigma_u, double
sigma_v, double orientation)

int standard_gabor(IMAGE *out, double radial_bandw, double fcentral,
double angular_bandw, double orientation)

int gabor_bank(IMAGE *out, double radial_bandw, double angular_bandw,
int nr)
```

***DESCRIPTION***

make_gabor() calculates an even-symmetric Gabor-filter in the Fourier-domain around the central frequency "fcentral" with standard deviations "sigma_u" and "sigma_v". The filter is rotated "orientation" degrees. For the function standard_gabor(), the filter sigmas can be specified as the radial bandwith "radial_bandw" (in octaves) and the angular bandwith "angular_bandw" (in degrees). gabor_bank() calculates a bank of standard_gabor filters for which the central frequencies are chosen in such way that the half-values of the filters are touching. The number of circles filled in the frequency domain is given by "nr".

***RETURN VALUES***

IE_OK (1) on succes
Negative error status on failure (see im_error.h)

***SEE ALSO***

apply_frequency_bank

## *make_image*

***NAME***

make_image - make an image with a display window attached

***SYNOPSIS***

```
#include "im2scil.h"

IMAGE *make_image(char *name, int type, int lenx, int leny, int lenz,
int posx, int posy)
```

***DESCRIPTION***

make_image() creates an image with the specified "name" of the specified "type" with the specified dimensions "lenx", "leny" and "lenz". make_image() also creates a display window for the image at ("posx","posy"). The available standard image types (see image.h) are:

GREY_2D
BINARY_2D
FLOAT_2D
COMPLEX_2D
GREY_3D
BINARY_3D
FLOAT_3D
COMPLEX_3D
COLOR_2D
COLOR_3D
LABEL_2D
LABEL_3D

make_image() works by calling create_image() followed by create_display().

***RETURN VALUES***

A pointer to the newly defined image
NULL if the image could not be created.

***SEE ALSO***

create_image  create_display  destroy_image  roi_define

## *MakeNewMenu*

## *AddToMenu*

## *ActivateMenu*

***NAME***

MakeNewMenu - building block for system independent menu model

AddToMenu - building block for system independent menu model

ActivateMenu - make menu visible

***SYNOPSIS***

```
#include "md_gen.h"

ABSTRACT *MakeNewMenu(char *name, ABSTRACT *parentmenu)

void AddToMenu(char *name, ABSTRACT *menu, ABSTRACT *parentmenu)

void ActivateMenu(ABSTRACT *top)
```

***DESCRIPTION***

MakeNewMenu:

| | |
|---|---|
| name | Name of the new menu |
| parentmenu | Abstract pointer to parent menu. If NULL then a new menu tree is started |

AddToMenu:

| | |
|---|---|
| name | Name of menu item |
| menu | Indicates whether a pull-right menu or just an item should be added. If NULL name is seen as an simple item, otherwise name is the name of the pull-right item in the parent menu |
| parentmenu | Parent menu to which the item or pull-right should be added. |

ActivateMenu:

| | |
|---|---|
| top | The root of a menu tree |

MakeNewMenu() and AddToMenu() are the two building blocks with which any menu can be build. Any Menu and Dialog generator for SCIL must contain these two routines. The system independent parts of SCIL uses these two routines to build its menus.

A menu is activated or made invisible through:
ActivateMenu(top);

***EXAMPLE***

The simple interface will consist of:

```
ABSTRACT *FillMenu(TopLevel, menustr, ... menustr, NULL)
ABSTRACT *MenuFromFile(file)
      Abstract Menu model
```

```
                      +--------+
Basic building block: |  label | Item/submenu name
                      +--------+
                      |   ptr  | NULL or pointer to submenu
                      +--------+
```

Example menu:

```
   File                          Option
+----------+                  +----------+
| Special ---->+-------+      | Special ------>+-------+
| open     |   | spec1 |      | dialog   |     | spec1 |
| close    |   | spec2 |      | menu     |     | spec2 |
| quit     |   +-------+      | spec2    |     +-------+
+----------+                  +----------+
```

Abstract model:

```
Menu handle
           +--------+
           | Control|
           +---+----+
               |
               V
           <---+---->  <---+---->
               |           |
               V           V
           +--------+  +--------+
           |  File  |  | Option |
           +---+----+  +---+----+
               |           |
               |           V
               |       <---+---> <----+---> <---+--> <---+--->
               |           |          |         |        |
       +-----~ | ~---------+          V         V        V
       |       |                 +--------+ +------+ +-------+
       |       |                 | dialog | | menu | | spec2 |
       |       |                 +---0----+ +---0--+ +---0---+
       |       V
       |   <---+----> <---+---> <----+---> <---+--->
       |       |          |          |          |
       |       V          V          V          V
       |   +--------+ +-------+ +--------+ +-------+
       +-> | Special| | open  | | close  | | quit  |
           +---+----+ +---0---+ +---0----+ +---0---+
               |
               |
               V
           <---+---> <---+--->
               |         |
               V         V
          +--------+ +-------+
          | spec1  | | spec2 |
          +---0----+ +---0---+
```

The example menu can be build by:

```
THING Menu, File, Option, Special;

/* Start new menu tree */
Menu = MakeNewMenu("Control", NULL);
/* Create File menu and add to top */
File = MakeNewMenu("File", Menu);
/* Create Option menu and add to top */
```

```
Option = MakeNewMenu("Option", Menu);
/* Create Special menu,add to File menu*/
Special = MakeNewMenu("Special", File);

AddToMenu("open", 0, File);  /* Add open item to File menu */
AddToMenu("close", 0, File); /* Add close item to File menu */
AddToMenu("quit", 0, File);  /* Add quit item to File menu */

/* Add Special menu to Option menu */
AddToMenu("Special", Special, Option);

AddToMenu("dialog", 0, Option); /* Add dialog item to Option menu */
AddToMenu("menu", 0, Option);   /* Add menu item to Option menu */
AddToMenu("spec2", 0, Option);  /* Add spec2 item to Option menu */

AddToMenu("spec1", 0, Special); /* Add spec1 item to Special menu */
AddToMenu("spec2", 0, Special); /* Add spec2 item to Special menu */
```

***NOTE***

Although the routines can be used directly in a SCIL session, they are meant to be used by system programmers only. For occasional use of menu’s, other more easy to use menu creation interfaces will be provided for. These interfaces however do use these routines to build the menu’s.

***RETURN VALUES***

MakeNewMenu() returns a pointer to an abstract structure representing the new menu.

AddToMenu() and ActivateMenu return nothing

## *malloc*

## *free*

## *realloc*

## *calloc*

***NAME***

malloc, free, realloc, calloc - main memory allocator

***SYNOPSIS***

```
void *malloc(unsigned int size)

void free(void *ptr)

void *realloc(void *ptr, unsigned int size)

void *calloc(unsigned int nelem, unsigned int elsize)
```

***DESCRIPTION***

These functions are interface functions to the standard C functions as implemented on the current operating system. The functionality of these functions is:

malloc() and free() provide a simple general-purpose memory allocation package. malloc() returns a pointer to a block of at least size bytes beginning on a word boundary.

The argument to free() is a pointer to a block previously allocated by malloc(); this space is made available for further allocation, but its contents are left undisturbed.

Needless to say, grave disorder will result if the space assigned by malloc() is overrun or if some random number is handed to free().

malloc() allocates the first big enough contiguous reach of free space found in a circular search from the last block allocated or freed, coalescing adjacent free blocks as it searches. It calls sbrk (see break(2)) to get more memory from the system when there is no suitable space already free.

realloc() changes the size of the block pointed to by "ptr" to "size" bytes and returns a pointer to the (possibly moved) block. The contents will be unchanged up to the lesser of the new and old sizes.

calloc() allocates space for an array of "nelem" elements of size "elsize". The space is initialized to zeros.

Each of the allocation routines returns a pointer to space suitably aligned (after possible pointer coercion) for storage of any type of object.

***RETURN VALUES***

malloc, realloc and calloc return a null pointer (0) if there is no available memory or if the area has been detectably corrupted by storing outside the bounds of a block. malloc may be recompiled to check the area very stringently on every transaction; see the source code.

---

## *max_element*

## *min_element*

***NAME***

max_element - determine position of maximum

min_element - determine position of minimum

***SYNOPSIS***

```
#include "im_proto.h"

int max_element(IMAGE *in, VAR_OBJECT *result, int whole, int
dimension)

int min_element(IMAGE *in, VAR_OBJECT *result, int whole, int
dimension)
```

***DESCRIPTION***

Determine the position of the pixel with minimum/maximum value in image "in" and store the result in the var_object "result". If "whole" is set (=1) then the position of the minimum/maximum of the entire image will be determined. If "whole" is not specified (=0) then the position of the minimum/maximum of each line is determined. In that case the following has to taken into account: If "dimension" is 1 then the minimum/maximum each x-line is determined. (2: each y-line, 3: each z-line if a 3d-image).

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

pix_minval  pix_maxval  pix_average_val

---

## *maxelm*

***NAME***

maxelm

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See max_element

---

## *maximum_cost_path*

***NAME***

maximum_cost_path - find an optimal path in a cost matrix

***SYNOPSIS***

```
#include "grey_2dp.h"

int maximum_cost_path(IMAGE *input, VAR_OBJECT *output, int markov,
int circular)
```

***DESCRIPTION***

Using a dynamic programming technique, find an 8-connected path connecting the top and bottom boundaries of the image "input". Of all possible paths, the path is chosen with the maximum sum over its pixels. "markov" (default 0) is added to the pixels in the current search direction, a value assigned to it decreases the probability that the path becomes overly curved.

The algorithm is such that each horizontal line of the image "input" contains precisely one pixel on the found path. These horizontal positions are stored in the 1-dimensional array "output". Pixels in "input" with a negative value are forbidden points; the algorithm will find a path without using these points.

If the "circular" is Yes (1), "input" is treated "circularly", i.e. the pixels in the last row of "input" are assumed to be the upper neighbors of the pixels in the first row. Specifying this restricts the algorithm to solutions with 8-connectivity between the path’s points in the first and the last row of the image "input".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

maximum_trace resample_perp back_project drawcurve

---

## *maximum_im*

***NAME***

maximum_im - element wise maximum

***SYNOPSIS***

```
#include "im_proto.h"

int maximum_im(IMAGE *in1, IMAGE *in2, IMAGE *out)
```

***DESCRIPTION***

Compare each element of "in1" with the corresponding element of "in2", take the maximum value of the two and store this value in the corresponding element of "out".

***NOTE***

For more powerful image arithmetic expressions (scaling, adding offsets, etc.), use the function eval().

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

minimum_im

## *maximum_trace*

***NAME***

maximum_trace - find a path tracing maximum value

***SYNOPSIS***

```
#include "grey_2dp.h"

int maximum_trace(IMAGE *input, IMAGE *output, int startx, int
starty, int dir, int avglen, int length, int minedge, double minval,
VAR_OBJECT *table)
```

***DESCRIPTION***

Starting at position ("startx", "starty") in the image "input", the command searches for the direction in which the mean grey-value is maximum. The initial search direction is specified by the parameter "dir", which may vary from 0 to 7, representing the following directions:

```
3  2  1
4  *  0
5  6  7
```

Given a current pixel (x,y) and a current direction (d), there are  three candidate points for the next pixel, viz. the neighbor points  in the directions (d-1), (d) and (d+1) (modulo 8). To decide which  to take, three straight lines are considered, one in the direction (d),  another in between the directions (d) and (d+1) and a third in between the directions (d) and (d-1). Along each of these lines the mean value is calculated, over a length of "avglen" pixels. The next pixel is the candidate point that corresponds with the direction with the maximum average. The process is repeated at the new pixel, using  the found direction as a new value for (d).

The process stops:

- If the distance of the candidate pixel  to the image boundaries is less than the value of "minedge".
- If the maximum of the mean values in the candidate directions is less than the specified value "minval".
- If a pixel is obtained that is set in the binary image "output
- If the number of pixels found becomes equal to the value "length".

The coordinates x and y of the i-th pixel obtained are stored into the locations (0,i) and (1,i) of the var_object "table" of type SHORT_T. The table "table" is optional; it should be specified if the found path is to  be used by command resample_perp(). The pixels found are set in the binary image "output". Except for  offering a way to stop the search as mentioned above, this image  is only for inspection. For further processing of the path,  operand "table" should be used.

If the VAR_OBJECT "table" is specified, the number of points obtained in the trace is equal to the length of the VAR_OBJECT "table"  (1st dimension) after the operation; "table" is adjusted to the  correct length.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***
maximum_cost_path  resample_perp  back_project  drawcurve

---

## ***maxval***

***NAME***
maxval

***DESCRIPTION***
This is an old function name, only provided for backward compatibility with TCL_Image routines.

See pix_maxval

---

## *measure*

### *NAME*

measure - high level object measurements

### *SYNOPSIS*

```
#include "im_aio.h"

LIST *measure(IMAGE *grey, IMAGE *binary, int garb, int inter,
unsigned long shape, unsigned long dens, int print_it, char *file)
```

### *DESCRIPTION*

| | | |
|---|---|---|
| grey | - | Grey value image containing original object |
| binary | - | Binary image containing mask of the objects |
| garb | - | Object garbage level |
| inter | - | Interactive mode (disabled) |
| print_it | - | Print results |
| shape, dens | - | Bitmaps with feature specification |
| file | - | Store results in file |

measure() is the highest level measurement routine of the AIO package.

The function list_label() is used to label the objects in the binary image using 8 connectivity and a garbage level "garb". Then the functions object_shape_meas() and object_dens_meas() are used to measure the shape and densitometry features specfied in the "shape" and "dens" bitmaps. The results of the measurements are shown on your terminal/worksheet if "print_it" is 1. If a filename other than "-" is given in "file" the results are also stored in that file.

### *NOTE*

As of version 2.0 of the Image library, the interaction switch "inter" has been disabled. The routine only performs automatic measurement of the all objects in the binary image. In SCIL_Image, a new interactive function Imeasure() has been introduced to perform the interactive measurement of objects.

### *RETURN VALUES*

A list with object information is returned on success
NULL on failure.

### *SEE ALSO*

object_shape_meas  object_dens_meas  list_label

## ***memchr***

## ***memcmp***

## ***memcpy***

## ***memmove***

## ***memset***

***NAME***

memchr, memcmp, memcpy, memmove, memset - memory operations

***SYNOPSIS***

```
#include <string.h>

void *memchr(void *s, int c, int n)

int memcmp(void *s, void *t, int n)

void *memcpy(void *s, void *t, int n)

void *memmove(void *s, void *t, int n)

void *memset(void *s, int c, int n)
```

***DESCRIPTION***

This function is an interface to the standard C function as implemented on the current system. The functionality of this function is:

memchr() returns a pointer to the first occurrence if character "c" in memory area "s" or NULL if "c" is not present among the first "n" characters.

memcmp() compares the first "n" characters of memory area "s" with memory area "t"; it returns <0 if "s" < "t"; 0 if "s" == "t", or >0 if "s" > "t".

memcpy() copies "n" characters from memory area "t" to memory area "s", and returns "s"

memmove() copies "n" characters from memory area "t" to memory area "s", Copying between areas that overlap is handled correctly. memmove() returns "s"

memset() places character "c" into the first "n" characters of memory area "s", it returns "s"

***RETURN VALUES***

see the description of each of the functions

***SEE ALSO***

strchr   strcmp  strcpy

## *merge*

***NAME***

merge - image merge

***SYNOPSIS***

```
#include "im_proto.h"

int merge(IMAGE *in, IMAGE *out, int direct, int iter)
```

***DESCRIPTION***

Merge two halves of image "in" storing the lines of one half into even positions and the other half into odd positions of the image "out". The command is executed on a per row or per column basis as specified by "direct", "direct" = 0 means merging horizontal lines, "direct" = 1 means merging vertical lines and "direct" = 3 means merging Z-slices (if "in" is a 3D image). The command is repeated "iter" times.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

merge_horizontal  merge_vertical  split

---

## *merge_horizontal*

***NAME***

merge_horizontal - merge in horizontal direction

***SYNOPSIS***

```
#include "im_proto.h"

int merge_horizontal(IMAGE *in, IMAGE *out, int iter)
```

***DESCRIPTION***

Same as merge() with "direct" = 0.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

merge_vertical  merge  split

---

## *merge_vertical*

***NAME***

merge_vertical - merge in vertical direction

***SYNOPSIS***

```
#include "im_proto.h"

int merge_vertical(IMAGE *in, IMAGE *out, int iter)
```

***DESCRIPTION***

Same as merge() with "direct" = 1.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

merge_horizontal  merge  split

---

## *mergh*

***NAME***

mergh

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See merge_horizontal

---

## *mergv*

***NAME***

mergv

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See merge_vertical

---

## *message_line_info*

***NAME***

message_line_info - show textual info to the user

***SYNOPSIS***

```
#include "md_gen.h"

int message_line_info(int code, char *buf)
```

***DESCRIPTION***

message_line_info() create a window that can be used to show textual messages to the user. The message can be transferred in lines to the window module with the message_line_info(). The total message for the window can contain a maximum of 10,000 bytes. If the window is too small to show the whole text the user can scroll the text up and down.

The "code" argument can be used to define a block-structure in the  text, however nothing is yet done with the block information, but may be done with it in the future. It must however be used to indicate the end of the text (code = -2).

"code" indicates :

| | |
|---|---|
| 0: | Start of possible more than one block;  "text" is ignored. |
| 1: | Start of text-block. |
| 2: | .. further lines of the text-block. |
| -1: | End of text-block |
| -2: | End of all text-blocks, show message window; "text" is ignored. |

The programmer is responsible for the placing the newlines in the text. The routines does not place any newlines in the text.

***EXAMPLE***

```
/* example to show text information to the user */
message_line_info(0,"This text is ignored");
message_line_info(1,"Textual information for the user:\n\n");
message_line_info(2,"This is the info you wanted to pass\n");
message_line_info(2,"Rest of the textbody..\n");
message_line_info(2,"Even more lines\n");
message_line_info(-1,"Last text line.\n");
message_line_info(-2,"This text is also ignored");
```

***RETURN VALUES***

| | |
|---|---|
| OK (1) | on success |
| NOT_OK (0) | when the maximum number of characters has been passed |

## *minelm*

***NAME***

minelm

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See min_element

---

## *minimum_im*

***NAME***

minimum_im - element wise minimum

***SYNOPSIS***

```
#include "im_proto.h"

int minimum_im(IMAGE *in1, IMAGE *in2, IMAGE *out)
```

***DESCRIPTION***

Compare each element of "in1" with the corresponding element of "in2", take the minimum value of the two and store this value in the corresponding element of "out".

***NOTE***

For more powerful image arithmetic expressions (scaling, adding offsets, etc.), use the function eval().

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

maximum_im

---

## *minval*

***NAME***

minval

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See pix_minval

---

## *mirrh*

***NAME***

mirrh

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See mirror_horizontal

---

## *mirror*

***NAME***

mirror - mirror image

***SYNOPSIS***

```
#include "im_proto.h"

int mirror(IMAGE *in, IMAGE *out, int direct)
```

***DESCRIPTION***

Mirror image "in" relative to a central axis and store the result in "out". The direction of the axis may be horizontal or vertical as specified by "direct". The command is executed on a per row or per column basis as specified by "direct", "direct" = 0 means a horizontal mirror is used (top-line becomes bottom line), "direct" = 1 means a vertical mirror and "direct" = 3 means a Z-mirror is used (if "in" is a 3D image).

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

mirror_horizontal  mirror_vertical  rotate

---

## *mirror_horizontal*

***NAME***

mirror_horizontal - mirror in horizontal direction

***SYNOPSIS***

```
#include "im_proto.h"

int mirror_horizontal(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

Same as mirror() with "direct" is 0.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

mirror_vertical mirror rotate

---

## *mirror_vertical*

***NAME***

mirror_vertical - mirror in vertical direction

***SYNOPSIS***

```
#include "im_proto.h"

int mirror_vertical(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

Same as mirror() with "direct" is 1.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

mirror_horizontal mirror rotate

---

## *mirrv*

***NAME***

mirrv

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See mirror_vertical

---

## *mix*

***NAME***

mix - pixel wise compare and select new pixel value

***SYNOPSIS***

```
#include "im_proto.h"

int mix(IMAGE *in, IMAGE *out, long thres, long val1, long val2, long
val3)
```

***DESCRIPTION***

Compare each pixel in the image "in" with a test-value "thres" and select a new pixel out of a choice of three, following the rules:

- if the input pixel is less than the "thres", the new pixel value will be "val1"

- if the input pixel is equal to the "thres", the new pixel value will be "val2"

- if the input pixel is greater than the "thres", the new pixel value will be "val3"

The resulting pixels are stored into the image "out".

The values "thres", "val1", "val2" and "val3" can either be scalars or images, in any combination. If the value is a scalar the value is of course a constant value. If the value is an image then the value of the pixel that corresponds with the pixel in the input image is taken.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

tri_state_threshold clip  threshold  contrast_stretch equalize  lookup

---

## *mix_filter*

***NAME***

mix_filter - get sum and difference of two filters

***SYNOPSIS***

```
#include "im_proto.h"

int mix_filter(IMAGE *in1, IMAGE *in2, IMAGE *sum, IMAGE *diff, IMAGE
*alpha)
```

***DESCRIPTION***

Stores the weighted sum and difference filters of "in1" and "in2" according to:

sum = alpha*in1 + (1.-alpha)*in2;
diff = alpha*in1 - (1.-alpha)*in2;

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

random_filter

## *mmops3x3*

***NAME***

mmops3x3 - mathemaical morphological operators

***SYNOPSIS***

```
#include "im_proto.h"

unsigned long mmops3x3(IMAGE *X, IMAGE *Y, IMAGE *M, int S, int T,
int opcode, int edge)
```

***DESCRIPTION***

The function mmops3x3() basically performs the hit-or-miss transform (HOM) on the images "X" and "Y" using structuring elements "S" and "T" restricted to a 3x3 neighborhood. The result is stored in the image "Y". The result of the HOM may be combined with a mask image "M". The operation performed depends on the the parameter "opcode". The "edge" parameter specifies whether to set (1) or clear (0) the border around the image before the operation.

Notation (Morphology for programmers)

| | |
|---|---|
| $X \otimes [S,T]$ | : Hit-or-miss transform using structuring elements S (object) and T (background). |
| $X^c$ | : Complement of X |
| $X \cap Y$ | : Intersection (AND) of X and Y |
| $X \cup Y$ | : Union (OR) of X and Y |
| $X \setminus Y$ | : Difference ( $X \setminus Y = X \cap Y^c$ ) |

| Opcode (symbolic) | (numeric) | Operation |
|---|---|---|
| HOM | 1 | $Y = X \otimes [S,T]$ |
| THINN | 2 | $Y = X \bigcirc [S,T] == X \setminus ( X \otimes [S,T] )$ |
| THICK | 3 | $Y = X \odot [S,T] == X \cup ( X \otimes [S,T] )$ |
| C_HOM | 4 | $Y = M \cap ( X \otimes [S,T] )$ |
| C_THINN | 5 | $Y = M \cap ( X \bigcirc [S,T] )$ |
| C_THICK | 6 | $Y = M \cap ( X \odot [S,T] )$ |
| C_NOTHOM | 7 | $Y = M \cap ( X \otimes [S,T] )^c$ |
| NOT_CHOM | 8 | $Y = ( M \cap (X \otimes [S,T] )^c$ |
| NOTC_HOM | 9 | $Y = M^c \cap ( X \otimes [S,T] )$ |
| NOT_HOM | 10 | $Y = ( X \otimes [S,T] )^c$ |
| NOTC_NOTHOM | 11 | $Y = M^c \cap ( X \otimes [S,T] )^c$ |

Structuring Element:

The structuring elements S and T (both restricted to the 3x3 square) are coded with a decimal number with 3 digits. Each digit encodes one of the three rows:

```
4  2  1

x  x  x          4+2+1 = 7
.  x  .            2   = 2
.  .  .                = 0    coded as 720
```

Second example:

```
4  2  1

.  .  .                = 0
.  .  .                = 0
x  x  x          4+2+1 = 7    coded as 7
```

Don’t type in leading zeros in the encoding of a structuring element as in C this means an otcal number. This will lead to unexpected results (but not what you wanted).

A short introduction

The simplest morphological operations are the erosion and dilation. The erosion of a binary image X with a structuring element S is denoted as

$$X \ominus S$$

and it selects from the input image all those pixels where the structuring element S fits within the object X. That is, at each position in the image we check whether all neighbours as defined with the set S are object pixels in the original (i.e. have binary value 1). If so then that pixel is part of the eroded set. If not then the pixel is a background pixel.

The hit-or-miss transform is the intersection (AND) of two erosions. The object set (the original image) is eroded with structuring element S and the background set (the inverted original image) is eroded with background set T. The intersection of these two erosions results is the hit-or-miss transform (opcode HOM):

$$X \otimes [S,T] = X \ominus S \cap X^c \ominus T$$

This means that a pixel at position (i,j) in an image is an object pixel (binary value 1) in the hit-or-miss result in case

all neighbour pixels indicated with the object set S are object pixels in the original image

AND

all neighbour pixels indicated with the background set T are background pixels in the the original image

Please note that this implies that the structuring sets S and T should have no pixels in common (unless you want to clear an image...).

As an example consider the hit-or-miss transform using the following structuring sets:

```
        . . .          x x x
    S = . x .      T = x . x
        . . .          x x x
```

Note that the dots are placeholders. They are not part of the set, they are indicated just to "see" the entire 3x3 neighbourhood. An other way of looking at this is to say that they are "don’t care" pixels. Note that because S and T should have no pixels in common we denote this as

```
               o o o
      [S,T] =  o x o
               o o o
```

where elements of S are denoted with "x" and elements of T with "o". The hit-or-miss transform of an image with this mask [S,T] will detect all isolated pixels in the original image. The isolated pixels in a binary image can be removed by taking the difference of the above hit-or-miss transform with the original image:

$X \bigcirc [S,T] = X \setminus X \otimes [S,T]$

this operation is called "thinning" and its opcode is THINN.

The "dual" operator of thinning is thickening (opcode THICK). Now we don’t remove the pixels found by the hit-or-miss transform but instead we "add" them to the image:

$X \odot [S,T] = X \cup X \otimes [S,T]$

The results of hit-or-miss transforms, thinnings and thickenings can be combined with a mask image M leading to the opcodes C_HOM (conditional HOM), C_THINN etc. This can be useful in several algorithms.

***EXAMPLE***

1) Finding the object contour

The (8 connected) object contour is found by taking the difference of the original set and the eroded set (erosion using the 4 connected neighbourhood as structuring element):

```
        . x .
    S = x x x
        . x .
```

$X \setminus X \ominus S$

Although the erosion is not directly available in mmops3x3 we can easily implement the above by taking

```
        . . .
    T = . . .   (no pixels at all)
```

```
          . . .
```

and then

$$X \setminus X \ominus S = X \setminus X \otimes [S,T]$$

so the command to perform this operation is:

```
      mmops3x3 in out out 272 0 THINN 0
```

2) Connectivity preserving thinning

A skeleton operation can be interpreted as connectivity preserving thinning: as long as the connectivity is not broken, pixels from the contour of a set are removed. This is the essence of the skeleton algorithms based on thinning (as opposed to skeletons based on maximal disks). A very simple implementation of this idea is the following piece of C-code that indeed implements something of a skeleton (but the "built-in" skeletons are much better!):

```
#include "image.h"


thinskelet(in, out)
IMAGE *in, *out;
{
      IMAGE *tmp;
      int n;

      tmp = make_image "tmp";
      copy_im in out;

      do {
            copy_im out tmp;

            n = mmops3x3 tmp out out 720   7 THINN;
            n = mmops3x3 out out out 660  13 THINN;
            n = mmops3x3 out out out 464 111 THINN;
            n = mmops3x3 out out out  66 310 THINN;
            n = mmops3x3 out out out  27 700 THINN;
            n = mmops3x3 out out out  33 640 THINN;
            n = mmops3x3 out out out 131 444 THINN;
            n = mmops3x3 out out out 330  46 THINN;

      } while( !equal_images(tmp, out) );

      destroy_image tmp;
}
```

***RETURN VALUES***

The function returns the number of pixels in the resultant image:
retval = N( $X\otimes(S,T)$ ) on success
IE_NOT_OK (0)  on failure

## *modal_input*

***NAME***

modal_input - modal text input window

***PLATFORM***

Unix

***SYNOPSIS***

```
#include "md_gen.h"

int modal_input(char *buf0, …)
```

***DESCRIPTION***

modal_input() is a means to prompt the user for input, and waits until that input is given. Both textual input a well as a choice from a number of buttons (max. 8) is possible.

A window is popped up in which the specified strings are displayed and at the bottom one up to eight button are visible.

The text, the buttons and the input field are all specified as strings (char pointers), but they must comply with the following rules:

1) A button string must be present in which the buttons are defined between square brackets ([buttontext]). The maximum number of buttons is eight. Only one button string is allowed.

2) A maximum of 16 strings is possible of which the last must be the button string. In a string the newline character (’\n’) may be used to accommodate more lines.

3) The input field buffers must be specified after the button strings, the maximum number of input fields is eight. An input field is specified by the keyword "INPUT" in the text strings, no other text may be present in that string. Each input field must be a existing character buffer of sufficient length.

Below are a few examples to illustrate their use.

***EXAMPLE***

```
/* example 1; without text input */
int choice;

choice = modal_input("SCIL_Image is beautiful\n", "[Yes][No]");
if (choice == 1) printf("We think so too !\n");
if (choice == 2) printf("But we think it is !\n");
/* end of example 1 */

/* example 2; with text input */
int choice;
char buf[100];

sprintf(buf, "this is the default text");
choice = modal_input("What do you think of SCIL_Image ?",
              "INPUT",
              "(type your answer above\nit will appear also in image
A)",
              "[OK]", buf);
printf(" Your answer was --%s--\n",buf);
disp_text(a,0,100,buf);
/* end of example 2 */
```

***RETURN VALUES***

The number of the button pressed

---

## *modulo_im*

***NAME***

modulo_im - modulo (remainder of integer division)

***SYNOPSIS***

```
#include "im_proto.h"

int modulo_im(IMAGE *in1, IMAGE *in2, IMAGE *out)
```

***DESCRIPTION***

Divide each element of "in1" by the corresponding element of "in2" and store the remainder of the division in the corresponding element of "out".

***NOTE***

For more powerful image arithmetic expressions (scaling, adding offsets, etc.), use the function eval().

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

div_im

---

## *MouseMove*

***NAME***

MouseMove - test whether the mouse moved with a button down

***SYNOPSIS***

```
#include "disp_p.h"

int MouseMove(IM_EVENT mouse_event)
```

***DESCRIPTION***

MouseMove() can be used to find out whether the mouse moved with one or more buttons down. The function returns either TRUE or FALSE. MouseMove() can only be used after a call to the "point_im()" routine which returns a mouse-event as one of its arguments.

***EXAMPLE***

```
#include "disp_p.h"
#include "image.h"

IMAGE    *ip;
int       x, y;
IM_EVENT  event;

while (point_im(&ip, &x, &y, &event) != ’q’) {
      if (MousePress(event))
            printf("Don’t move.\n");
      else if(MouseMove(event))
            printf("I said, don’t move\n");
}
```

***RETURN VALUES***

| | |
|---|---|
| TRUE (non zero) | if moved |
| FALSE (zero) | otherwise |

***SEE ALSO***

point_im  MousePress  MouseRelease  IsMouseDown  EventType  KeyPressed

## *MousePress*

***NAME***

MousePress - test whether a button has been pressed

***SYNOPSIS***

```
#include "disp_p.h"

int MousePress(IM_EVENT mouse_event)
```

***DESCRIPTION***

MousePress() can be used to find out whether a mouse button has been pressed. Depending on the button pressed the function returns one of the symbolic values LEFT, MIDDLE, RIGHT defined in imwindow.h. MousePress() can only be used after a call to the "point_im" routine which returns a "mouse-event" as one of its arguments.

***EXAMPLE***

```
#include "disp_p.h"
#include "image.h"

IMAGE    *ip;
int       x, y;
IM_EVENT  event;

while (point_im(&ip, &x, &y, &event) != ’q’)
      if (MousePress(event) == LEFT)
            disp_vector(ip, 0, 0, x, y);
```

***RETURN VALUES***

| | |
|---|---|
| MIDDLE, LEFT, RIGHT | if a button was pressed |
| 0 | if no button was pressed |

***SEE ALSO***

point_im  MouseRelease  MouseMove  IsMouseDown  EventType  KeyPressed

## *MouseRelease*

***NAME***

MouseRelease - test whether a button has been released

***SYNOPSIS***

```
#include "disp_p.h"

int MouseRelease(IM_EVENT mouse_event)
```

***DESCRIPTION***

MouseRelease() can be used to find out whether a mouse button has been released. Depending on the button released the function returns one of the symbolic values LEFT, MIDDLE, RIGHT defined in imwindow.h. MouseRelease() can only be used after a call to the "point_im" routine which returns a "mouse-event" as one of its arguments.

***EXAMPLE***

```
#include "disp_p.h"
#include "image.h"

IMAGE    *ip;
int       ox, oy;
int       x, y;
IM_EVENT  event;

disp_draw_mode;
while (point_im(&ip, &x, &y, &event) != ’q’) {
      if (MousePress(event) == LEFT){
            ox = x; oy = y;
            disp_rect(ip, ox, oy, 20, 20);
      }
      else if(!MouseRelease(event) == LEFT){
            disp_rect(ip,ox,oy,20,20);
            disp_rect(ip,x,y,20,20);
            ox = x; oy = y;
      }
}
```

***RETURN VALUES***

| | |
|---|---|
| MIDDLE, LEFT, RIGHT | if a button was released |
| 0 | if no button was released |

***SEE ALSO***

point_im  MousePress  MouseMove  IsMouseDown  EventType  KeyPressed

## *muj*

***NAME***

muj

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See conjugate_mul_im

---

## *natural_window_size*

***NAME***

natural_window_size - give an image a window of its own size

***SYNOPSIS***

```
#include "disp_p.h"

int natural_window_size(IMAGE *im)
```

***DESCRIPTION***

This function adjusts the size of the window of an image to the size of the image.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

set_window_size  set_window_pos  set_start_pos

---

## *nearest_int*

***NAME***

nearest_int - truncate pixel values to nearest integer value

***SYNOPSIS***

```
#include "im_proto.h"

int nearest_int(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

Convert each element of image "in" to an integer value by means of rounding and store the result into the corresponding elements of image "out". The effect is that the integer value most close to the original value is taken. If the fractional part is exactly 0.5, for positive values the integer value just greater than the original value is taken and for negative values the integer value just less than the original value is taken.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

fraction_im  truncate_im  lowest_int

---

## *negation_im*

***NAME***

negation_im - negation

***SYNOPSIS***

```
#include "im_proto.h"

int negation_im(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

Negate each element of "in" and store the result in the corresponding element if "out".

***NOTE***

For more powerful image arithmetic expressions (scaling, adding offsets, etc.), use the function eval() .

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

sign_im  eval

---

## next_plane

***NAME***

next_plane - display next plane or slice of an image

***SYNOPSIS***

```
#include "disp_p.h"

int next_plane(IMAGE *im, int num)
```

***DESCRIPTION***

Display the next or previous slice or plane of an image. "num" == "-1" takes the previous slice, "0" takes the current slice and "1" takes the next slice. See also "auto_plane".

***RETURN VALUES***

None

***SEE ALSO***

display_image  set_display_slice  auto_plane

---

## object_contour

***NAME***

object_contour - transform object contour into freeman chain code

***SYNOPSIS***

```
#include "im_aio.h"

LIST *object_contour(IMAGE *mask, LIST *link)
```

***DESCRIPTION***

| | | |
|---|---|---|
| mask | - | Image containing labeled objects |
| link | - | Link pointing to object information |

AIO primitive to obtain the freeman code representation of an objects contour.

***NOTE***

object_contour() is part of the AIO package, and is meant to be only visible at the programming level.

***RETURN VALUES***

A list with the freeman code
NULL on failure

***SEE ALSO***

object_shape_meas

---

## *object_dens_meas*

***NAME***

object_dens_meas - densitometry measurements on a labeled object

***SYNOPSIS***

```
#include "im_aio.h"

int object_dens_meas(IMAGE *grey, IMAGE *mask, LIST *link, unsigned
long bitmap)
```

***DESCRIPTION***

| | | |
|---|---|---|
| grey | - | Image containing original grey value information |
| mask | - | Image containing labeled objects. |
| link | - | Link pointing to object information |
| bitmap | - | Bitmap with feature specification |

object_dens_meas() is the function to measure any densitometry features of an object. (Also used by the higher level measure() routine)

In the bitmap GREYVAL, TRANSMIS and OD can be specified or a (logical OR) combination off these values.

The features measured are:

mean value
integrated sum
standard deviation

The results are stored together with the other object information. Results can be obtained through functions:

grey_mean(object)
trans_mean(object)
od_mean(object)
grey_sum(object)
trans_sum(object)
od_sum(object)
grey_stdev(object)
trans_stdev(object)
od_stdev(object)

***EXAMPLE***

```
To measure the optical density parameters of all the objects in a
list:

#include "image.h"
#include "im_aio.h"
LIST *l, *o;

readfile("cermet",a,0,0);
threshold(a,b,128);
invert_im(b,b);
l = list_label(b, c, 8, 0);
FORALL(o,l){
      object_dens_meas(a, c, o, OD);
      printf("Mean %f Sum %f Stdev %f\n", od_mean(o), od_sum(o),
      od_stdev(o));
}
```

***NOTE***

object_dens_meas() is part of the AIO package

***RETURN VALUES***

IE_OK (1) if measurement was successful
Negative error status on failure (see im_error.h)

***SEE ALSO***

measure  object_shape_meas  grey_mean trans_mean  od_mean
grey_sum  trans_sum  od_sum  grey_stdev  trans_stdev  od_stdev

## *object_freeman_meas*

***NAME***

object_freeman_meas - object measurement of features based on freeman chain of contour

***SYNOPSIS***

```
#include "im_aio.h"

int object_freeman_meas(IMAGE *mask, LIST *link, unsigned long
bitmap)
```

***DESCRIPTION***

| | | |
|---|---|---|
| mask | - | Image containing labeled objects. |
| link | - | Link pointing to object information |
| bitmap | - | Bitmap with feature specification |

AIO primitive to measure shape features based on freeman chaincode of the contour of objects. Used by the higher level object_shape_meas() routine. In the bitmap the following features or a combination (logical OR) of them can be specified:

PERI
CR
BEND

The results are stored together with the other object information. Results can be obtained through functions as in:

peri(object)
cr(object)
bend(object)

***NOTE***

object_freeman_meas() is part of the AIO package, and is meant to be only visible at the programming level.

***RETURN VALUES***

IE_OK (1) if measurement was successful
Negative error status on failure (see im_error.h)

***SEE ALSO***

measure object_shape_meas object_dens_meas peri cr bend

## *object_moment_meas*

***NAME***

object_moment_meas - object measurement of features based on moments

***SYNOPSIS***

```
#include "im_aio.h"

int object_moment_meas(IMAGE *mask, LIST *link, unsigned long bitmap)
```

***DESCRIPTION***

| | | |
|---|---|---|
| mask | - | Image containing labeled objects. |
| link | - | Link pointing to object information |
| bitmap | - | Bitmap with feature specification |

AIO primitive to measure shape features based on moments. Used by the higher level object_shape_meas() routine. In the bitmap the following features or a combination (logical OR) of them can be specified:

GRAVX
GRAVY
ANGLE
ECCENTR
LAXIS
SAXIS

The results are stored together with the other object information. Results can be obtained through functions as in:

gravx(object)
gravy(object)
angle(object)
eccentr(object)
laxis(object)
saxis(object)

***NOTE***

object_moment_meas() is part of the AIO package, and is meant to be only visible at the programming level.

***RETURN VALUES***

IE_OK (1) if measurement was successful
Negative error status on failure (see im_error.h)

***SEE ALSO***

measure  object_shape_meas  object_dens_meas
angle  eccentr  gravx  gravy  laxis  saxis

## *object_rect_to_silo*

***NAME***

object_rect_to_silo - add object from image to an image-silo

***SYNOPSIS***

```
#include "image.h"
#include "silo.h"

int object_rect_to_silo(SILOPTR siloptr, int silo_key, IMAGE
*srcimage, LIST *link)
```

***DESCRIPTION***

| | | |
|---|---|---|
| siloptr | - | Pointer to the image-silo. |
| silo_key | - | Numerical entry in the image-silo. |
| srcimage | - | Source image. |
| link | - | Link pointing to object |

Copies the rectangle enclosing the object pointed to by "link"  from the image "srcimage" to the silo "siloptr" at the position "silo_key".

***RETURN VALUES***

IE_OK (1) on succes
Negative error status on failure (see im_error.h)

## *object_shape_meas*

***NAME***

object_shape_meas - shape measurements on a labeled object

***SYNOPSIS***

```
#include "im_aio.h"

int object_shape_meas(IMAGE *mask, LIST *link, unsigned long bitmap)
```

***DESCRIPTION***

| | | |
|---|---|---|
| mask | - | Image containing labeled objects. |
| link | - | Link pointing to object information |
| bitmap | - | Bitmap with feature specification |

object_shape_meas() is the function to measure any shape feature of an object. (Also used by the higher level measure() routine) The results are stored together with the other object information.

| Currently implemented: | Results can be obtained through functions as in: |
|---|---|
| AREA | area(object) |
| PERI | peri(object) |
| CR | cr(object) |
| BEND | bend(object) |
| GRAVX | gravx(object) |
| GRAVY | gravy(object) |
| ANGLE | angle(object) |
| XMIN | xmin(object) |
| XMAX | xmax(object) |
| YMIN | ymin(object) |
| YMAX | ymax(object) |
| WIDTH | width(object) |
| HEIGHT | height(object) |
| LAXIS | laxis(object) |
| SAXIS | saxis(object) |
| ECCENTR | eccentr(object) |

***EXAMPLE***

```
To measure the area, perimeter, center of gravity and contour ratio
of all the objects in a list:

#include "image.h "
#include "im_aio.h"
LIST *l, *o;

readfile("cermet",a,0,0);
threshold(a,b,128);
invert_im(b,b);
l = list_label(b, c, 8, 0);
FORALL(o,l){
      object_shape_meas(c, o, AREA|PERI|GRAVX|GRAVY|CR);
      printf("Area %d Perimeter %f\n", area(o), peri(o));
      printf("Center of gravity (%d,%d), Contour Ratio %f\n",
            gravx(o), gravy(o), cr(o));
}
```

***NOTE***

object_shape_meas() is part of the AIO package

***RETURN VALUES***

IE_OK (1) if measurement was successful
Negative error status on failure (see im_error.h)

***SEE ALSO***

measure  object_dens_meas
area  peri  cr  bend  gravx  gravy  angle  xmin  xmax  ymin  ymax
width  height  laxis  saxis  eccentr

---

## *objectsize*

***NAME***

objectsize - object size estimation

***SYNOPSIS***

```
#include "im_proto.h"

int objectsize(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

Calculate for each object in the labeled image "in" the object size by counting the number of pixels, assign to all pixels of the object the calculated size and store the result in image "out". So for each object the label number is replaced by the object size.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

hull  label  rhull  small_object_removal

---

## *odd_fsizes_ok*

***NAME***

odd_fsizes_ok - check if filter sizes are odd and in the range

***SYNOPSIS***

```
#include "im_infra.h"

int odd_fsizes_ok(int fx, int fy, int fmax)
```

***DESCRIPTION***

The functions checks if the arguments "fx" and "fy" are odd values and in the range from one to "fmax". The range is checked by the function range_ok() and the check if the values are odd by the function odd_ok(). If one of the values is outside the range an error is generated and the following message is added to the error-stack:

```
Filter Width [<fx>] out of range (1..<fmax>)
```

or

```
Filter Height [<fy>] out of range (1..<fmax>)
```

When one of the values is not an odd value then the message will be:

```
Filter Width [<fx>] should be odd
```

or

```
Filter Height [<fy>] should be odd
```

***RETURN VALUES***

IE_OK (1) if the values "fx" and "fy" are O.K.
IE_NOT_OK (0) if either of the values is even or out of the range.

***SEE ALSO***

range_ok  odd_ok

## *odd_ok*

***NAME***

odd_ok - check if a value is a odd integer value

***SYNOPSIS***

```
#include "im_infra.h"

int odd_ok(int value, char *text)
```

***DESCRIPTION***

The parameter "value" is checked to see if it is an odd value. If it is not an odd value an error is generated and the following message is added to the error-stack:

```
<text> [<value>] should be odd
```

***RETURN VALUES***

IE_OK (1) if the value is odd
IE_NOT_OK (0) if the value is even

***SEE ALSO***

even_ok  odd_fsizes_ok

## *open*

***NAME***

open - open for reading or writing

***SYNOPSIS***

```
int open(char *name, int mode)
```

***DESCRIPTION***

This function is an interface to the standard C function as implemented on the current system. The functionality of this function is:

open() opens the file "name" for reading (if "mode" is 0), writing (if "mode" is 1) or for both reading and writing (if "mode" is 2). "name" is the address of a string of ASCII characters representing a pathname, terminated by a null character.

The file is positioned at the beginning (byte 0). The returned file descriptor must be used for subsequent calls for other input-output functions on the file.

***RETURN VALUES***

The value -1 is returned if the file does not exist, if one of the necessary directories does not exist or is unreadable, if the file is not readable (resp. writable), or if too many files are open.

***SEE ALSO***

creat  read  write  close

---

## *open_silo*

***NAME***

open_silo - open an existing image-silo

***SYNOPSIS***

```
#include "silo.h"

SILOPTR open_silo(char *siloname)
```

***DESCRIPTION***

| | | |
|---|---|---|
| siloname | - | filename of the image-silo. |

Opens a silo by the name "siloname". Reads in all the entries to make an internal entry list. Returns a handle to this silo which must be handed to all silo-I/O functions.

***RETURN VALUES***

A pointer to the silo structure
NULL if an error occurred.

---

## *opening3x3*

***NAME***

opening3x3 - open

***SYNOPSIS***

```
#include "im_proto.h"

int opening3x3(IMAGE *in, IMAGE *out, int iter, int con, int bound)
```

***DESCRIPTION***

Performs an opening from "in" to "out", which is performed by "iter" erosions from "in" to "out" followed by "iter" dilations from "out" to "out". The operation deletes objects having a width less than two times the specified number of cycles. "bound" specifies that the edge around the image must be set to foreground (1) or to background (0) pixels.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

closing3x3  erosion3x3  dilation3x3

---

## *or_im*

***NAME***

or_im - bitwise or of images

***SYNOPSIS***

```
#include "im_proto.h"

int or_im(IMAGE *in1, IMAGE *in2, IMAGE *out)
```

***DESCRIPTION***

Perform a bitwise OR operation of each element of "in1" with the corresponding element of "in2" and store the result in "out"

***NOTE***

For more powerful image arithmetic expressions (scaling, adding offsets, etc.), use the function eval().

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

and_im  xor_im  invert_im  shift_im

---

## *overload_func*

## *overloadable_func*

## *type_overload_func*

## *type_overloadable_func*

***NAME***

overload_func - get overloaded function pointer

overloadable_func - check if a function is overloadable

type_overload_func - get overloaded function pointer for a specified image type

type_overloadable_func - check if a function is overloadable for a specified image type

***SYNOPSIS***

```
#include "im_infra.h"

FPI overload_func(char *name, IMAGE *im)

FPI type_overload_func(char *name, int type)

FPI overloadable_func(char *name, IMAGE *im)

FPI type_overloadable_func(char *name, int type)
```

***DESCRIPTION***

overload_func() and type_overload_func() check if the generic function "name" is overloadable for the specific image "im" or image-type "type". If it is, they return the C-function pointer to the overloaded function. If it is not overloadable, they generate an error and return a NULL pointer.

type_overloadable_func() and overloadable_func() perform the same functionality but do NOT generate an error if the function is not overloadable.

***RETURN VALUES***

A pointer to the function
NULL if not overloadable.

***SEE ALSO***

init_func_overload  show_func_overload

## *palette2color*

***NAME***

palette2color - convert a palette image into a RGB color-image

***SYNOPSIS***

```
#include "im_proto.h"

int palette2color(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

palette2color() converts the palette-image "in" into a full-color RGB image in image "out" A palette-image is a grey-value image with a color-lookup-table attached in which a limited numbers of colors is present with which each of the grey-value values is displayed. Because the grey-values serve only as an index in the lookup-table, no sensible image-processing can be done on the image. Converting the image to a full-color RGB image, makes image-processing possible again.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

set_clut

## *parabolic_dilation*

## *parabolic_erosion*

## *parabolic_opening*

## *parabolic_closing*

***NAME***

parabolic_dilation - grey value dilation using a parabolic structuring function

parabolic_erosion - grey value erosion using a parabolic structuring function

parabolic_opening - grey value closing using a parabolic structuring function

parabolic_closing - grey value opening using a parabolic structuring function

***SYNOPSIS***

```
#include "im_proto.h"

int parabolic_dilation(IMAGE *in, IMAGE *out, double rho)

int parabolic_erosion(IMAGE *in, IMAGE *out, double rho)

int parabolic_closing(IMAGE *in, IMAGE *out, double rho)

int parabolic_opening(IMAGE *in, IMAGE *out, double rho)
```

***DESCRIPTION***

These functions perform the morphological dilation, erosion, closing and opening on the grey value image "in" using a parabolic structuring function

q(x,y) = 1/(4 rho) (x^2 + y^2)

and stores the result in "out". The "width" of the parabola is given by the "rho" parameter. The larger "rho" the wider the parabola becomes and the larges the effective neighborhood size is.

***EXAMPLE***

1) Background correction

The following code fragment performs a closing on the "schema" image to wipe out the black drawing. The result of the closing is an approximation of the white paper. Taking the difference between closing and original (black tophat) gives the drawing in white on a black background.

```
int i;
readf schema
parabolic_clos a b 2
grey_morph_round a c 23 Yes CLOSE
i=90;profile(a,i);profile(b,i);profile(c,i);
```

Note that both closings do the job. Both are isotropic (rotational symmetric structuring functions). The parabolic closing is faster though and the resulting function is smoother.

2) Euclidean distance transform

The parabolic erosion can be used to calculate the Euclidean distance transform of a binary image. This is illustrated with the following code fragment:

```
set_display_mode c LIN_STRETCH YES Yes;
set_display_mode d LIN_STRETCH YES Yes;

eval a=(xx==128&&yy==128)?255:0;
parabolic_dila a b 5;
thres b b 1;

distance b c 5 7 12;
/* note that this is 5 times the distance */

binary_to_grey b d 10000;
parabolic_erosion d d 0.25;
/* note that 4*0.25 = 1 ==> erosion with ’unit’ parabola */

eval d=irint(5*sqrt(d));
```

Note that in the Chamfer distance transform (in image "C") the octagonal shape of the distance cone can be readily seen. If one is interested in the differential geometrical structure of the distance transform then the Euclidean one should be prefered. However in case one is interested in the constrained distance transform, the Chamfer distance is the best solution.

***LITERATURE***

R.v.d.Boomgaard, L.Dorst, S. Makram-Ebeid, J. Schavemaker, "Quadratic structuring functions in mathematical morphology", in "Mathematical morphology and its applications to image and signal processing", (eds. P.Maragos, R.W. Schafer and M.A. Butt), Kluwer Adademic Publishers, 1996, pp. 147---154.

***RETURN VALUES***

IE_OK(1) on success
IE_NOT_OK (0) on failure (see im_error.h)

## *part_from_silo*

***NAME***

part_from_silo - transfer an image from silo to part of an image.

***SYNOPSIS***

```
#include "image.h"
#include "silo.h"

int part_from_silo(SILOPTR siloptr, int silo_key, IMAGE *dstimage,
int left, int top)
```

***DESCRIPTION***

| | | |
|---|---|---|
| siloptr | - | Pointer to the image-silo. |
| silo_key | - | Numerical entry silo_key. |
| dstimage | - | Numerical image in which the part must fit. |
| left | - | Start x-coordinate of the part-image. |
| top | - | Start y-coordinate of the part-image. |

Copies the "silo_key" image from the image-silo "siloptr" and places this in image "dstimage" at coordinate "left", "top".

***RETURN VALUES***

IE_OK (1) on succes
Negative error status on failure (see im_error.h)

---

## *part_image_display*

***NAME***

part_image_display - display part of an image

***SYNOPSIS***

```
#include "im2scil.h"

int part_image_display(IMAGE *im, int sx, int sy, int sz, int width,
int height, int depth)
```

***DESCRIPTION***

part_image_display() displays the part of image "im" with dimensions "width"*"height"*"depth" located at position ("sx","sy","sz").

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

display_image

---

## *part_to_silo*

***NAME***

part_to_silo - add part of image to an image-silo

***SYNOPSIS***

```
#include "image.h"
#include "silo.h"

int part_to_silo(SILOPTR siloptr, int silo_key, IMAGE *srcimage, int
left, int top, int sizex, int sizey)
```

***DESCRIPTION***

| | | |
|---|---|---|
| siloptr | - | Pointer to the image-silo. |
| silo_key | - | Numerical entry in the image-silo. |
| srcimage | - | Source image. |
| left | - | Start x-coordinate of the part-image. |
| top | - | Start y-coordinate of the part-image. |
| sizex | - | Width of the part-image. |
| sizey | - | Height of the part-image. |

Copies the part with sizes "sizex" * "sizey" at the position ("left", "top") from the image "srcimage" to the silo "siloptr" at the position "silo_key" in the silo.

***RETURN VALUES***

IE_OK (1) on succes
Negative error status on failure (see im_error.h)

## *percentile*

***NAME***

percentile - percentile filtering

***SYNOPSIS***

```
#include "im_proto.h"

int percentile(IMAGE *in, IMAGE *out, int fx, int fy, int num)
```

***DESCRIPTION***

Perform a percentile filter, a generalization of the median filter. Image "in" is scanned with a moving window with dimensions "fx" * "fy". The pixel values within this window are sorted. The pixel which has (after sorting) sequence number "num" (starting from "1") is taken as the output pixel value and is stored in the pixel in image "out" that corresponds with the central pixel of the window.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

## *peri*

***NAME***

peri - obtain perimeter of an object

***SYNOPSIS***

```
#include "im_aio.h"

double peri(LIST *link)
```

***DESCRIPTION***

| | | |
|---|---|---|
| link | - | Link pointing to object |

AIO primitive to obtain value of an object feature

peri() returns the perimeter of the object pointed to by "link" if this has previously been measured.

***RETURN VALUES***

perimeter of object on success
0.0 if link is not an object or if angle has not been measured.

***SEE ALSO***

measure  object_shape_meas  object_dens_meas

## *perror*

## *errno*

***NAME***

perror, errno - system error messages

***SYNOPSIS***

```
void perror(char *s)

int sys_nerr;

char *sys_errlist[];

int errno;
```

***DESCRIPTION***

This function is an interface to the standard C function as implemented on the current system. The functionality of this function is:

perror() produces a short error message on the standard error describing the last error encountered during a call to a system or library function. If "s" is not a NULL pointer and does not point to an empty string, the string it points to is printed, followed by a colon, followed by a space, followed by the message and a NEWLINE. If "s" is a NULL pointer or points to an empty string, just the message is printed, followed by a NEWLINE. To be of most use, the argument string should include the name of the program that incurred the error. The error number is taken from the external variable errno (see intro(2)), which is set when errors occur but not cleared when non-erroneous calls are made.

To simplify variant formatting of messages, the vector of message strings "sys_errlist" is provided; "errno" can be used as an index in this table to get the message string without the newline. "sys_nerr" is the number of messages provided for in the table; it should be checked because new error codes may be added to the system before they are added to the table.

## *phase_im*

***NAME***

phase_im - find phase of every complex image element

***SYNOPSIS***

```
#include "im_proto.h"

int phase_im(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

For each complex element a + bi of "in" the phase atan2(b,a) is returned in the range [-pi,pi] and stored in the image "out". If "out" is a complex image then the result will be stored in the real part of each element of "out" and the imaginary part will be cleared, otherwise the image "out" will be a float image.

***RETURN VALUES***

Number of domain conflicts (number of would be atan2(0,0)) so 0 is OK.
Negative error status (see im_error.h).

## *pix_abs_sum*

***NAME***

pix_abs_sum - returns absolute sum of pixels

***SYNOPSIS***

```
#include "im_proto.h"

double pix_abs_sum(IMAGE *in)
```

***DESCRIPTION***

Add all absolute values of the elements of image "in" and returns the result.

***RETURN VALUES***

The absolute sum.

***SEE ALSO***

abs_im  pix_sum

## *pix_average_val*

***NAME***

pix_average_val - calculate the average value of all pixels

***SYNOPSIS***

```
#include "im_proto.h"

int pix_average_val(IMAGE *in, VAR_OBJECT *result, int whole, int
dimension)
```

***DESCRIPTION***

Calculate the average pixel value of the image "in" and store the result in the object "result". If "whole" is set (=1) the average value of the entire image will be calculated. If "whole" is not set (=0) then the average value is calculated on a line-by-line basis. In that case the following has to be taken into account: If "dimension" is 1 then the average of each x-line is calculated. (2: each y-line, 3: each z-line if a 3D-image). The object "result" will be automatically adjusted so that all data will fit. If the object "result" is DONT_STORE (a NULL pointer) then the result will be printed on the terminal(this is only possible when "whole" is set).

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

pix_minval  pix_maxval  max_element  min_element

## *pix_count*

***NAME***

pix_count - count pixels with a certain value

***SYNOPSIS***

```
#include "im_proto.h"

long pix_count(IMAGE *in, int val)
```

***DESCRIPTION***

Count the number of pixels in image "in" that have value "val".

***RETURN VALUES***

The number of pixels or
Negative error status on failure (see im_error.h)

## *pix_minval*

## *pix_maxval*

***NAME***

pix_minval - calculate minimum pixel value

pix_maxval - calculate maximum pixel value

***SYNOPSIS***

```
int pix_minval(IMAGE *in, VAR_OBJECT *result, int whole, int
dimension)

int pix_maxval(IMAGE *in, VAR_OBJECT *result, int whole, int
dimension)
```

***DESCRIPTION***

Calculate the minimum/maximum value of the image "in" and store the result in the object "result". If whole is specified (=1) the minimum/maximum value of the entire image will be calculated. If whole is not specified (=0) then the minimum/maximum value is calculated on a line-by-line basis. In that case the following has to be taken into account: If "dimension" is 1 then the minimum/maximum of all x-lines is calculated. (2: all y-lines, 3: all z-lines if a 3d-image). The object "result" will be automatically adjusted so that all data will fit in. If the object "result" is DONT_STORE (a NULL pointer) then the result will be printed on the terminal(this is only possible when "whole" is specified).

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

min_element  max_element  pix_average_val

## *pix_sum*

***NAME***

pix_sum - calculate the sum of the pixel values

***SYNOPSIS***

```
#include "im_proto.h"

int pix_sum(IMAGE *in, VAR_OBJECT *sum)
```

***DESCRIPTION***

Calculate the sum of all pixel values within the image "in", and store that sum in the var_object "sum" if an object is specified. In case of a NULL pointer (DONT_STORE) the result will be printed on the terminal.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

---

## *pix_value_str*

***NAME***

pix_value_str - get the value of a "pixel" in a string

***SYNOPSIS***

```
#include "im_proto.h"

char *pix_value_str(IMAGE *im, int x, int y, int z)
```

***DESCRIPTION***

pix_value_str() returns the value of the pixel located at ("x","y","z") in the image "im" in string format.  The buffer in which the string is stored is a global char buffer of length 100.

***RETURN VALUES***

Pointer to the global buffer in which the string is stored.

---

## *pixval*

***NAME***

pixval - pixel value of an object

***SYNOPSIS***

```
#include "im_aio.h"

PIXEL pixval(LIST *link)
```

***DESCRIPTION***

| | | |
|---|---|---|
| link | - | Link pointing to the object |

AIO primitive to obtain value of an object feature

pixval() returns the pixel value (label) of the object pointed to by "link".

This feature need not be specified beforehand as it is automatically measured during the labeling process.

***RETURN VALUES***

The pixel value (label) of the object on success
0 if link is not an object

***SEE ALSO***

measure  object_shape_meas  object_dens_meas  list_label

## *pl_io_ok*

***NAME***

pl_io_ok - check if the specified bitplanes are in the correct range

***SYNOPSIS***

```
#include "im_infra.h"

int pl_io_ok(int in, int out)
```

***DESCRIPTION***

This function is meant to check if the values "in" and "out" are valid bitplanes of grey valued images. The function performs a call to range_ok() for each of the two values. The valid range for the bitplanes is specified by the defines "MIN_PLANE" (1) and "MAX_PLANE" (16) from the include file "image.h ". If "in" or "out" is out of range an error is generated and one of the following messages is added to the error-stack:

```
Input bitplane [<in>] out of range (1..16).  (for the "in"
parameter)

Output bitplane [<out>] out of range (1..16). (for the "out"
parameter)
```

***RETURN VALUES***

IE_OK (1) if the value are in the range
IE_NOT_OK (0) if either of the values is outside the range

***SEE ALSO***

plane_ok  range_ok

## *plane_ok*

***NAME***

plane_ok - check if the plane is in the correct range.

***SYNOPSIS***

```
#include "im_infra.h"

int plane_ok(int plane)
```

***DESCRIPTION***

The bitplane "plane" is checked if it is in the range of valid bitplane of the grey-valued images. This range is determined by the defines "MIN_PLANE" (1) and "MAX_PLANE" (16) in the file "image.h ". The function calls range_ok() to check the range. If "plane" is outside the correct range an error is generated and the following message is added to the error-stack:

```
      Bitplane [<plane>] out of range (1..16)
```

***RETURN VALUES***

IE_OK (1) if the plane is in the correct range.
IE_NOT_OK (0) if it is not

***SEE ALSO***

pl_io_ok  range_ok

## *plane_to_binary*

***NAME***

plane_to_binary - convert a grey value image bitplane to a binary image

***SYNOPSIS***

```
#include "bin_2dp.h"

int plane_to_binary(IMAGE *in, int plane, IMAGE *out)
```

***DESCRIPTION***

The specified plane "plane" of the grey_2d image "in" is converted to a binary image "out"

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

binary_to_plane  binary_to_grey  set_im_type

## *planecopy*

***NAME***

planecopy - copy a bitplane between grey valued images

***SYNOPSIS***

```
#include "im_proto.h"

int planecopy(IMAGE *in, int inplane, IMAGE *out, int outplane)
```

***DESCRIPTION***

Copy bitplane "inplane" from image "in", to bitplane "outplane" in image "out".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

copy_im  plane_to_binary  binary_to_plane

---

## *plot_histogram*

***NAME***

plot_histogram - plot histogram on the controlling terminal

***PLATFORM***

UNIX.

***SYNOPSIS***

```
#include "im2scil.h"

int plot_histogram(IMAGE *in, int action, int clip)
```

***DESCRIPTION***

Calculates the grey level histogram of image "in" and plot the resulting histogram on the controlling terminal. The grey level range of image "in" is subdivided into 64 classes or bins. "action" can be one of normal (0), cumulative (1), trunc_peak (2) or trunc_at_clip. "clip" is used for clipping.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

histogram  histdata  hist2d

---

## *point_im*

***NAME***

point_im - get information of mouse pointer in image

***SYNOPSIS***

```
#include "disp_p.h"

int point_im(IMAGE **imptr, int *xptr, int *yptr, int *butptr)
```

***DESCRIPTION***

point_im() returns either on a keyboard hit or when a button is pressed inside a display window of an image.

The x and y-coordinates are passed through "xptr", "yptr". Button information is passed through "butptr". The image in which the pointing event took place is passed through "imptr".

If a key was received first then all other information is not valid.

***RETURN VALUES***

ASCII code of the key pressed
0 if an event happened inside the display window before keypress

***SEE ALSO***

MousePress MouseRelease MouseMove IsMouseDown EventType KeyPressed poll_mouse

## *point_im_display_buf*

***NAME***

point_im_display_buf - text buffer to be displayed in pop-up window

***PLATFORM***

UNIX, Macintosh.

***SYNOPSIS***

```
#include "disp_p.h"

int point_im_display_buf(char *buf, int follow)
```

***DESCRIPTION***

This function can be used to specify the string of text "buf" that can be displayed in a pop-up window in one of the images. If the "pim" is activated by "handle_pim", the pop-up window will pop up below the cursor (if "follow" is set to 1) or in the top of the image pointed at.

***EXAMPLE***

```
See the demo "my_point.c" in the standard demo directory.
```

***RETURN VALUES***

None

***SEE ALSO***

handle_pim  point_im

## *point_object*

***NAME***

point_object - interactive object selection

***SYNOPSIS***

```
#include "im_aio.h"

LIST *point_object(IMAGE *image, LIST *list)
```

***DESCRIPTION***

| | | |
|---|---|---|
| image | - | Image with labeled objects |
| list | - | List with all the objects |

AIO primitive to select an object inside a labeled image from a list interactively by pointing with the mouse at the object of choice.

The routine can also be stopped, returning NULL if the return key is hit.

***EXAMPLE***

```
To interactively point at an object and copy this object to another
image the following could be typed:

#include "image.h "
#include "im_aio.h"
LIST *l, *o;

readfile("cermet",a,0,0);
threshold(a,b,128);
invert_im(b,b);
l = list_label(b, c, 8, 0);

set_im_type(d, LABEL_2D);
while(o = point_object(c, l)){
      copy_object(c, d, o);
      display_image(d);
}
l = rm_list(l);
```

***RETURN VALUES***

The link of the pointed object.
NULL if the routine was stopped by hitting return.

## *poll_mouse*

***NAME***

poll_mouse - poll the mouse position and button state

***PLATFORM***

UNIX, Macintosh.

***SYNOPSIS***

```
#include "disp_p.h"

int poll_mouse(IMAGE *image, int *im_x, int *im_y, int *win_x, int
*win_y)
```

***DESCRIPTION***

This function returns the position of the mouse pointer in the display window of the image "image" and the state of the buttons of the mouse. The position of the mouse is returned in both image- and window-coordinates.

The image-coordinates are returned through the parameters "im_x" and "im_y", which must be pointers. The window-coordinates are returned through "win_x" and "win_y", also pointers. The state of the mouse button(s) is returned through the return value of the function. The button state is the bitwise or-value of LEFT, MIDDLE and RIGHT (from include file "imwindow.h"). So if no button was pressed when the mouse was polled the return value would be 0.

NOTE: poll_mouse() is a polling function which means that the position and button state are returned immediately. point_im() however does not return until a mouse button or a key is pressed.

***RETURN VALUES***

The button state

***SEE ALSO***

point_im  MousePress  MouseMove  MouseRelease

## *positive_ok*

***NAME***

positive_ok - check if a value is positive

***SYNOPSIS***

```
#include "im_infra.h"

int positive_ok(int value, char *text)
```

***DESCRIPTION***

The integer "value" is checked to see if it is positive or not. If the value is negative an error is generated and the following message is added to the error-stack:

```
<text> [<value>] must be positive
```

Zero is considered to be positive as well in this function, if a check must be performed on a value that may not be zero then the function greater0_ok() can be used.

***NOTE***

This function can only handle integer values, to check on floating point values, use the function fpositive_ok().

***RETURN VALUES***

IE_OK (1) if the value is positive (zero included)
IE_NOT_OK (0) if the value is negative

***SEE ALSO***

greater0_ok  fpositive_ok

## *post_op*

***NAME***

post_op - perform image housekeeping after an operation

***SYNOPSIS***

```
#include "im_infra.h"

int post_op(IMAGE *out)
```

***DESCRIPTION***

post_op() performs some image infrastructure housekeeping on image "out", keeping the integrity of the image data-structure, and displaying the output image if the display mode is on.

post_op() should be called at the end of an image processing in which the pre_op() routine (with mode ADJUST(_NIP)) has been used on the image "out".

***RETURN VALUES***

IE_OK (1) if successful
IE_NOT_OK (0) if failed

***SEE ALSO***

pre_op

---

## *power_im*

***NAME***

power_im - power raising

***SYNOPSIS***

```
#include "im_proto.h"

int power_im(IMAGE *in1, IMAGE *in2, IMAGE *out)
```

***DESCRIPTION***

Raise each element of "in1" to the power given by the corresponding element of "in2" and store the result in the corresponding element of "out"

***NOTE***

For more powerful image arithmetic expressions (scaling, adding offsets, etc.), use the function eval() .

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

---

## *power_of_2_ok*

***NAME***

power_of_2_ok - check if a value is a power of 2

***SYNOPSIS***

```
#include "im_infra.h"

int power_of_2_ok(int value, char *text)
```

***DESCRIPTION***

The function checks to see if "value" is power of 2. If it is not an error is generated and the following message is added to the error-stack:

```
      <text> [<value>] must be a power of 2
```

***RETURN VALUES***

IE_OK (1) if the value is a power of 2
IE_NOT_OK (0) if it is not

## *pre_op*

## *set_cross_dim*

***NAME***

pre_op - function called before each image processing operation

set_cross_dim - set special sizes for the pre_op command

***SYNOPSIS***

```
#include "im_infra.h"

int pre_op(IMAGE *first, IMAGE *second, int mode, int first_spec,
unsigned long sec_type)

void set_cross_dim(int cross_x, int cross_y, int cross_z)
```

***DESCRIPTION***

The pre_op() function is part of the image infrastructure. Depending upon the "mode" argument, pre_op() either checks input images ("mode" = COMPARE), or adjusts the output image(s) ("mode" is ADJUST or ADJUST_NIP). set_cross_dim() is used to specify special dimensions for the pre_op() function if it is to adjust the output image.

The pre_op() function must be called just before the low-level image processing function is called. Furthermore pre_op must always be used in combination with the post_op() routine.

In SCIL_Image images are arranged in a flat class hierarchy. In this hierarchy there can be 32 different virtual image main-classes, and an unlimited number of image sub-classes. Each type of image is a unique sub-class instance. The data layout of any sub-class image must physically match with its main-class image. The sub-class image, however, can define its own extra information elements and its own class specific operations. The main-class of the "first" image is specified through the so-called main class specifier "first_spec". The type or sub-class of the second image is specified through the "sec_type" argument.

The meaning of the main-class "first_spec" argument, and the "sec_type" argument depends upon the value of the mode argument.

MODES:

| | |
|---|---|
| COMPARE | pre_op() is called with the COMPARE mode when two input images must be checked to be of (a) certain main-class(es) in order for the low-level image processing function to operate properly.<br><br>The "first_spec" argument is used to specify what kind of image main-classes are allowed for the first image argument. All legal class specifiers should be or’ed. |

The "sec_type" argument is used to specify that the main class of the "second" image must be the same as the main class of the given "sec_type" type.

A special case here is the constant "OUT_AS_IN" meaning that the second image must have the same main-class as the first image.

| | |
|---|---|
| ADJUST | pre_op() is called with the ADJUST mode when the output image must be set to the proper type and dimensions. |

The "first_spec" argument is used to specify what kind of image main-classes are allowed for the first image in order for the low-level image processing function to operate properly.

The "sec_type" argument is used to specify the type the second image should be adjusted to. When adjusting the "second" image the dimensions of the first image are taken, unless a special flag "cross_dimensions" is set. If the cross_dimensions flag is set than the global variables "cross_x", "cross_y", and "cross_z" will be used. These can be set using the set_cross_dim() function.

A special case is when the "sec_type" argument is set to "OUT_AS_IN". In this case the "second" image is adjusted to the same type and dimensions as the "first" image.

| | |
|---|---|
| ADJUST_NIP | This mode shows equal behavior to ADJUST mode, except that if "first" and "second" are the same image, always a new piece of memory will be allocated for the output. This option is meant for operations that cannot be performed in place like neighborhood operations. With this option the input and output image for such operations may be the same. |

***EXAMPLE***

```
      Below some typical examples, and special cases are given.

      * One input, one output.

      - Input and output must be grey_2d.

            uniform(in, out)
            IMAGE *in, *out;
            {
                  if (!pre_op(in, out, ADJUST, G_2D_SPEC, OUT_AS_IN))
                        return(FALSE)
                  low_uniform(.....);
                  post_op(out);
            }

      - Input must be grey_2d output must be binary_2d.

            threshold(in, out)
            IMAGE *in, *out;
            {
```

```
            if (!pre_op(in, out, ADJUST, G_2D_SPEC, BINARY_2D))
                  return(FALSE);
            low_threshold(.....);
            post_op(out);
      }
```

* Two inputs, one output.

- Inputs and output must be grey 2d/3d, output becomes the same as the inputs.

```
      add(in1, in2, out)
      IMAGE *in1,*in2,*out;
      {
            if (!pre_op(in1, in2, COMPARE, G_2D_SPEC | G_3D_SPEC,
                             OUT_AS_IN))
                  return(FALSE);
            if (!pre_op(in1, out, ADJUST, ImageTypeSpec(in1),
OUT_AS_IN))
                  return(FALSE);
            low_add(....);
            post_op(out);
      }
```

- First input must be grey_2d, second input must be binary_2d, output must become binary_3d.

```
      strange_func(in1, in2, out)
      IMAGE *in1,*in2,*out;
      {
            if (!pre_op(in1, in2, COMPARE, G_2D_SPEC, BINARY_2D))
                  return(FALSE);
            if (!pre_op(in1, out, ADJUST, G_2D_SPEC, BINARY_3D))
                  return(FALSE);
            low_strange_func(.......)
            post_op(out);
      }
```

* Special output adjustment.

- No output dimension adjustment. To avoid adjustment of the dimensions but allow type adjustment, for example to the type of the input image, the following can be done:

```
      blow(in, out)
      IMAGE *in, *out;
      {
            if (!pre_op(in, in, COMPARE, G_2D_SPEC, OUT_AS_IN))
                  return(FALSE);
            if (!pre_op(out, out, ADJUST, ImageTypeSpec(out),
                        ImageTypeIdent(in)))
                  return(FALSE);
            low_blow(....);
            post_op(out);
      }
```

- Output dimensions specifically set.

```
      strange_func(in, out)
      IMAGE *in, *out;
      {
            set_cross_dim( 50, 31, 1);
            if (!pre_op(in, out, ADJUST, G_2D_SPEC, BINARY_2D))
                  return(FALSE);
            strange_func(....);
            post_op(out);
      }
```

***RETURN VALUES***

IE_OK (1) on success

IE_NOT_OK (0) on failure (images do not match, image could not be converted etc.)

***SEE ALSO***

post_op

---

## *prewd*

***NAME***

prewd

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See prewitt_diff

---

## *prewitt_diff*

***NAME***

prewitt_diff - Prewitt differential type edge detector

***SYNOPSIS***

```
#include "im_proto.h"

int prewitt_diff(IMAGE *in, IMAGE *out, int mode)
```

***DESCRIPTION***

Differential edge detection based upon the Prewitt operator. Within the moving window in image "in", with dimensions 3*3, the horizontal and vertical differential values are calculated by a convolution with the masks (horizontal respectively vertical):

```
  1   1   1      -1   0   1
  0   0   0      -1   0   1
 -1  -1  -1      -1   0   1
```

The output value is calculated from these convolutions, depending on the "mode" specified:

| | |
|---|---|
| sqrt (1) | the output value is the square root of the sum of the quadratic convolution results. |
| sum (0) | the output value is the sum of the absolute values of the convolution results |

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

roberts_diff  sobel_diff  laplace  prewitt_temp  kirsch_temp  robinson_temp

## *prewitt_temp*

***NAME***

prewitt_temp - edge detection filter

***SYNOPSIS***

```
#include "im_proto.h"

int prewitt_temp(IMAGE *in, IMAGE *out, IMAGE *direction, int flag)
```

***DESCRIPTION***

Template type edge detection based upon the Prewitt operator. Within the moving window in the image "in", with dimensions 3 * 3, eight convolutions with the following masks are calculated:

$$\begin{array}{cccc}
(0) & (1) & (2) & (3) \\
\begin{matrix} -1 & 1 & 1 \\ -1 & -2 & 1 \\ -1 & 1 & 1 \end{matrix} &
\begin{matrix} 1 & 1 & 1 \\ -1 & -2 & 1 \\ -1 & -1 & 1 \end{matrix} &
\begin{matrix} 1 & 1 & 1 \\ 1 & -2 & 1 \\ -1 & -1 & -1 \end{matrix} &
\begin{matrix} 1 & 1 & 1 \\ 1 & -2 & -1 \\ 1 & -1 & -1 \end{matrix} \\
\\
(4) & (5) & (6) & (7) \\
\begin{matrix} 1 & 1 & -1 \\ 1 & -2 & -1 \\ 1 & 1 & -1 \end{matrix} &
\begin{matrix} 1 & -1 & -1 \\ 1 & -2 & -1 \\ 1 & 1 & 1 \end{matrix} &
\begin{matrix} -1 & -1 & -1 \\ 1 & -2 & 1 \\ 1 & 1 & 1 \end{matrix} &
\begin{matrix} -1 & -1 & 1 \\ -1 & -2 & 1 \\ 1 & 1 & 1 \end{matrix}
\end{array}$$

The output value is the maximum of the results of all these convolutions. It is stored into "out", in the pixel corresponding with the central pixel of the window. The sequence number of the convolution mask with the maximum result is an estimate of the direction of the first derivative and it is stored in the image "direction", if this is specified ("flag" = 1).

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

laplace  kirsch_temp  robinson_temp  prewitt_diff  roberts_diff  sobel_diff

## ***prewt***

***NAME***

prewt

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See prewitt_temp

## *printf*

## *fprintf*

## *sprintf*

***NAME***

printf, fprintf, sprintf - formatted output conversion

***SYNOPSIS***

```
#include <stdio.h>

int printf(char *format, ...)

int fprintf(FILE *stream, char *format, ...)

int sprintf(char *s, char *format, ...)
```

***DESCRIPTION***

These functions are interface functions to the standard C functions as implemented on the current operating system. The functionality of these functions is:

printf() places output on the standard output stream stdout. fprintf() places output on the named output stream. sprintf() places "output" in the string "s", followed by the character '\0'.

Each of these functions converts, formats, and prints its arguments after the first under control of the first argument. The first argument is a character string which contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which causes conversion and printing of the next successive arg printf.

Each conversion specification is introduced by the character %. Following the %, there may be:

- an optional minus sign "-" which specifies left adjustment of the converted value in the indicated field;

- an optional digit string specifying a field width; if the converted value has fewer characters than the field width it will be blank-padded on the left (or right, if the left-adjustment indicator has been given) to make up the field width; if the field width begins with a zero, zero-padding will be done instead of blank-padding;

- an optional period "." which serves to separate the field width from the next digit string;

- an optional digit string specifying a precision which specifies the number of digits to appear after the decimal point, for e- and f-conversion, or the maximum number of characters to be printed from a string;

- the character l specifying that a following d, o, x, or u corresponds to a long integer arg. (A capitalized conversion code accomplishes the same thing.)

- a character which indicates the type of conversion to be applied.

A field width or precision may be "*" instead of a digit string. In this case an integer arg supplies the field width or precision.

The conversion characters and their meanings are:

| | |
|---|---|
| d o x | The integer arg is converted to decimal, octal, or hexa-decimal notation respectively. |
| f | The float or double arg is converted to decimal notation in the style "[-]ddd.ddd" where the number of d’s after the decimal point is equal to the precision specification for the argument. If the precision is missing, 6 digits are given; if the precision is explicitly 0, no digits and no decimal point are printed. |
| e | The float or double arg is converted in the style "[-]d.ddde+_dd" where there is one digit before the decimal point and the number after is equal to the precision specification for the argument; when the precision is missing, 6 digits are produced. |
| g | The float or double arg is printed in style d, in style f, or in style e, whichever gives full precision in minimum space. |
| c | The character arg is printed. Null characters are ignored. |
| s | Arg is taken to be a string (character pointer) and characters from the string are printed until a null character or until the number of characters indicated by the precision specification is reached; however if the precision is 0 or missing all characters up to a null are printed. |
| u | The unsigned integer arg is converted to decimal and printed (the result will be in the range 0 to 2**32-1 |
| % | Print a ’%’; no argument is converted. |

In no case does a non-existent or small field width cause truncation of a field; padding takes place only if the specified field width exceeds the actual width. Characters generated by printf are printed by putc.

***EXAMPLE***

To print a date and time in the form "Sunday, July 3, 10:02", where weekday and month are pointers to null-terminated strings:

```
printf("%s, %s %d, %02d:%02d", weekday, month, day, hour, min);
```

To print pi to 5 decimals:

```
    printf("pi = %.5f", 4*atan(1.0));
```

***SEE ALSO***

putc scanf ecvt

---

## *propagation*

***NAME***

propagation - propagation

***SYNOPSIS***

```
#include "im_proto.h"

int propagation(IMAGE *in, IMAGE *mask, IMAGE *out, int iter, int
conn, int edge)
```

***DESCRIPTION***

Performs propagation ("masked expansion") on objects in image "in", masked by the image "mask" and stores the result in image "out". The objects in "in" are interpreted as the kernels of larger objects ("mask objects"). These mask objects are specified by "mask". The algorithm is executed by a repeated expansion of the kernel objects with the condition that the resulting pixels stay within the borders of the mask objects.

The expansion may be executed only "iter" times, "iter" is 0 specifies that the process must be repeated until all kernels in the input image have fully expanded within the mask objects in the mask image.

"conn" specifies the connectivity of the propagation and can be either 4 or 8. "edge" specifies if the pixels outside the image are to be seen as object pixels ("edge" = 1) or as background pixels ("edge" = 0).

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

---

## *ps_head*

## *ps_image*

## *ps_tail*

***NAME***

ps_head - open a file for (encapsulated) postscript output

ps_image - put an image in a postscript file

ps_tail - close a postscript output file

***SYNOPSIS***

```
#include "im_proto.h"

int ps_head(char *filename, int papersize)

int ps_image(IMAGE *image, int orient, int unit, double xpos, double
ypos, double xsiz, double ysiz, int border, char *comment, int
textsize)

int ps_tail(void)
```

***DESCRIPTION***

These functions can be used to create a (Encapsulated) Postscript file that contains one or more images.

**First** ps_head() must be used to open the postscript-file.

**Secondly**, use ps_image() one or more times to put one or more images in the postscript file.

**Finally**, use ps_tail() to close the Postscript file. Now the file can be send to a PostScript printer ( e.g. using the UNIX "lpr" command) or used in a text as encapsulated postscript.

ps_head() creates a file with the name "filename" and puts in some Postscript header information. "papersize" specifies the size of paper by one of these values:

1 = A4
2 = US_LETTER
3 = A3
4 = A5

ps_image() dumps the data of "image" in the postscript file previously opened with ps_head(). "orient" determines the orientation of the image, either portrait (=0) or landscape (=1). "unit" specifies the unit of measurement for the parameters "xpos", "ypos", "xsiz" and "ysiz", values are:

1 = inches
2 = points
3 = centimeters

4 = millimeters

"xpos" and "ypos" determine the position of the image on paper relative to the top-left corner. "xsiz" and "ysiz" specify the size of the image on paper.

When "border" is set to 1, a box is drawn around the image, 0 is no box. "comment" is an optional text-string that can be put under the image as a caption. "textsize" is the pointsize of the caption, the font is Times-Roman.

ps_tail() cleans up after ps_image() and closes the postscript file. If this function is not called, the Postscript file is incomplete and will produce unpredictable result when sent to a printer or used in text as a Encapsulated Postscript file.

***EXAMPLE***

```
ps_head("example.eps", 1)  /* open A4 postscript file */

/*
 * put image A in Landscape, at position 5.5, 6.7 (cm) of top right
 * corner, the sizes are 10.0 by 10.0 (cm) and draw a box around
 * it. Also print "Image A" under the image, using 10 points font
 */
ps_image(A, 1, 3, 5.5, 6.7, 10.0, 10.0, 1, "Image A", 10);

/* ps_image() may be called again to put more images in the file */

ps_tail()  /* close file "example.eps" , MUST BE USED */
```

***SEE ALSO***

im1ps  im2ps  im3ps  im4ps

## *pseudo*

***NAME***

pseudo - pseudo grey value graphics

***SYNOPSIS***

```
#include "im_proto.h"

int pseudo(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

Create a binary image with a pseudo-grey value impression of the grey value image "in" and store the result into the bitplane image "out". Each pixel of image "in" is replaced by a 6 x 6 binary mask, in which the ratio between the number of black and white pixels is proportional to the original grey-value.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

dither  greduce

---

## *psremoval*

***NAME***

psremoval - pepper and salt removal

***SYNOPSIS***

```
#include "im_proto.h"

int psremoval(IMAGE *in, IMAGE *out, int bound)
```

***DESCRIPTION***

Performs the "pepper and salt removal" operation on image "in" and stores the result in image "out". The image is scanned by a moving window with dimensions 3*3. If the central pixel within the window is the only object pixel within the window, it becomes a background pixel (value 0). If the central pixel is the only background pixel within the window, it becomes an object pixel (value 1). This operation deletes singular pixels (either fore- or background). "bound" specifies that the edge around the image must be set to foreground (1) or to background (0) pixels.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

---

## *put*

***NAME***

put

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See tri_state_threshold

---

## *put_xy_into_image*

***NAME***

put_xy_into_image - set pixels, given their coordinates

***SYNOPSIS***

```
#include "grey_2dp.h"

int put_xy_into_image(VAR_OBJECT *input, IMAGE *output, int value)
```

***DESCRIPTION***

A 2-dimensional VAR_OBJECT "input" of type SHORT_T with size 2 or 3 as its first dimension, is used to set pixels in the image "output". Per row of the array, one pixel is set in the output image.

If "input" has a row-length (first dimension) of 2, the pixels found in "input" are set to "value" (the two elements of the row are taken as the x- and y-coordinates of the pixels). If "input" has a row-length of 3, the three elements of the row are taken as the x- and y-coordinates and  the grey-value of the pixel to be put into "output".

All pixels with coordinates not found in "input" are left unchanged. If a pair of coordinates is found more than once the average of the input grey-values is taken for this pixel.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

chaincode_to_xy  image_to_chaincode  chaincode_to_image

---

## *putc*

## *putchar*

## *fputc*

## *putw*

***NAME***

putc, putchar, fputc, putw - put character or word on a stream

***SYNOPSIS***

```
#include <stdio.h>

int putc(int c, FILE *stream)

int putchar(int c)

int fputc(int c, FILE *stream)

int putw(int w, FILE *stream)
```

***DESCRIPTION***

These functions are interface functions to the standard C functions as implemented on the current operating system. The functionality of these functions is:

putc() appends the character "c" to the named output "stream". It returns the character written.

putchar(c) is defined as putc(c, stdout).

fputc() behaves like putc(), but is a genuine function rather than a macro.

putw() appends word "w" to the output "stream". It returns the word written. putw() neither assumes nor causes special alignment in the file.

The standard stream stdout is normally buffered if and only if the output does not refer to a terminal; this default may be changed by setbuf. The standard stream stderr is by default unbuffered unconditionally, but use of freopen() (see fopen()) will cause it to become buffered; setbuf(), again, will set the state to whatever is desired. When an output stream is unbuffered information appears on the destination file or terminal as soon as written; when it is buffered many characters are saved up and written as a block. fflush() (see fclose()) may be used to force the block out early.

***BUGS***

Because it is implemented as a macro, putc treats a stream argument with side effects improperly. In particular "putc(c, *f++);" doesn’t work sensibly.

Errors can occur long after the call to putc.

***RETURN VALUES***

These functions return the constant EOF upon error. Since this is a good integer, ferror should be used to detect putw errors.

***SEE ALSO***

fopen fclose getc puts printf fread

---

## *puts*

## *fputs*

***NAME***

puts, fputs - put a string on a stream

***SYNOPSIS***

```
#include <stdio.h>

int puts(char *s)

int fputs(char *s, FILE *stream)
```

***DESCRIPTION***

These functions are interface functions to the standard C functions as implemented on the current operating system. The functionality of these functions is:

puts() copies the null-terminated string "s" to the standard output stream stdout and appends a newline character.

fputs() copies the null-terminated string "s" to the named output "stream".

Neither routine copies the terminal null character.

***SEE ALSO***

fopen gets putc printf ferror fread

---

## *qpix*

***NAME***

qpix - plot binary image on laser printer

***PLATFORM***

***UNIX.***

***SYNOPSIS***

```
#include "im_proto.h"

int qpix(IMAGE *in, char *fname, int zoom, int append)
```

***DESCRIPTION***

Create a file "fname" to represent a hard-copy of the binary image "in" on a laser printer of Digital’s LN03 type. The plot file is an ASCII file containing the plot information represented in the LN03 pixel format. The magnification factor of the plot is specified by the factor "zoom".
The parameter "append" specifies if the plot file is to be appended to an existing plot file or text file (1= append, 0= not append). Appending the plot to a text file enables the possibility of merging text and figures. If appending is not specified and the file "fname" already exists, the existing plot file will be overwritten.

***NOTE***

The default value of "zoom" is 1 which gives 28.4 pixels per millimeter.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

---

## *quit*

***NAME***

quit - quit SCIL_Image session gracefully

***SYNOPSIS***

```
void quit(void)
```

***DESCRIPTION***

quit() exits SCIL_Image gracefully, flushing all the buffers and closing all open files.
It is synonymous to "exit(1)"

***RETURN VALUES***

None

***SEE ALSO***

exit

---

## *raise_window*

## *lower_window*

## *iconify_window*

## *deiconify_window*

***NAME***

raise_window - pop an image-window to the foreground

lower_window - push an image-window to the background

iconify_window - iconify an image-window

deiconify_window - deiconify an image-window

***PLATFORM***

UNIX, Macintosh

***SYNOPSIS***

```
#include "disp_p.h"

int raise_window(IMAGE *im)

int lower_window(IMAGE *im)

int iconify_window(IMAGE *im)

int deiconify_window(IMAGE *im)
```

***DESCRIPTION***

raise_window(), lower_window(), iconify_window() and deiconify_window() perform window manipulation on the display window of an image. For images without a display window, like ROIs and images created with create_image() NO error is generated.

raise_window() pops the display window of image "im" to the foreground.

lower_window() pushes the display window of image "im" to the background.

iconify_window() iconifies the display window of image "im"

deiconify_window() deiconifies the display window of image "im"

***RETURN VALUES***

IE_OK (1)

## *rand*

## *srand*

***NAME***

rand, srand - random number generator

***SYNOPSIS***

```
void srand(unsigned int seed)

int rand(void)
```

***DESCRIPTION***

These functions are interface functions to the standard C functions as implemented on the current operating system. The functionality of these functions is:

rand() uses a multiplicative congruential random number generator with period 2**32 to return successive pseudo-random numbers in the range from 0 to 2**31-1.

The generator is reinitialized by calling srand() with 1 as argument. It can be set to a random starting point by calling srand() with whatever you like as argument.

---

## *random_filter*

***NAME***

random_filter - make random filter image

***SYNOPSIS***

```
#include "im_proto.h"

int random_filter(IMAGE *out, double mean, double max, int symmetric)
```

***DESCRIPTION***

Creates a random filter with mean given by "mean" and maximum element given by "max". If symmetric is true (set to 1), the filter will be constrained by f(x,y) = f(-x,y).

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

random_im

---

## random_im

## srandom_im

***NAME***

random_im - fill an image with random values

srandom_im - set seed for random_im random generator

***SYNOPSIS***

```
#include "im_proto.h"

int random_im(IMAGE *im, int alt)

void srandom_im(long seed)
```

***DESCRIPTION***

random_im() fills the image "im" with random values. Two random generators are available. The "Normal" ("alt" = 0) can be influenced by a seed that can be set by srandom_im(). The "Alternate" ("alt" = 1) random generator cannot be externally influenced.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

eval

## *range_ok*

***NAME***

range_ok - check is a value is in the specified range

***SYNOPSIS***

```
#include "im_infra.h"

int range_ok(int value, int vmin, int vmax, char *text)
```

***DESCRIPTION***

range_ok() checks to see if "value" is in the range specified by "vmin" and "vmax" (borders included). If it is, a true status is returned. If "value" is outside the range then an error is generated and the following message is added to the error-stack:

```
<text> [<value>] out of range (<vmin>..<vmax>)
```

A lot of the checking routines use this function to do the actual checking and supply a default message for that specific check.

***EXAMPLE***

```
if range_ok() is called as "range_ok(18, 1, 9, "Number below 10");"
then the on the error-stack will be:

      Number below 10 [18] out of range (1..9)
```

***NOTE***

This function has the same functionality as frange_ok(). range_ok() can handle only integer values and frange_ok() can handle only floating point values.

***RETURN VALUES***

IE_OK (1) if the value is inside the range (borders included).
NOT_OK (0) if the value is outside the range.

***SEE ALSO***

frange_ok

## *raster*

***NAME***

raster - rasterization of an image

***SYNOPSIS***

```
#include "im_proto.h"

int raster(IMAGE *in, IMAGE *out, int factor, int ratio)
```

***DESCRIPTION***

Rasterise an image as follows: replace each pixel in the image "in" by a block of "factor" * "factor" pixels, the upper left "ratio" * "ratio" pixels having the original pixel value, and the remaining pixels having the background value 0. The result is stored into the image "out", which should at least be "factor" times bigger in both the x- and y-direction.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

## *read*

***NAME***

read - read from file

***SYNOPSIS***

```
int read(int fildes, char *buffer, int nbytes)
```

***DESCRIPTION***

These functions are interface functions to the standard C functions as implemented on the current operating system. The functionality of these functions is:

A file descriptor is a word returned from a successful open(), creat(), dup(), or pipe() call. "buffer" is the location of "nbytes" contiguous bytes into which the input will be placed. It is not guaranteed that all "nbytes" bytes will be read; for example if the file refers to a typewriter at most one line will be returned. In any event the number of characters read is returned.

If the returned value is 0, then end-of-file has been reached.

***RETURN VALUES***

As mentioned, 0 is returned when the end of the file has been reached. If the read was otherwise unsuccessful the return value is -1. Many conditions can generate an error: physical I/O errors, bad buffer address, preposterous nbytes, file descriptor not that of an input file.

***SEE ALSO***

open  creat

## ***read_var_object***

***NAME***

read_var_object - reads a var_object from a file

***SYNOPSIS***

```
#include "objectsp.h"

int read_var_object(char *filename, VAR_OBJECT *object)
```

***DESCRIPTION***

"read_var_object" reads the var_object specified by the pointer "object" from a file.
When executed, this function looks for two files on disk. A file with the name "filename".voh and a file the name "filename".vod.
The file with the extension ".voh" is an ASCII header file which describes the var_object. In the file with the extension ".vod" the actual data resides.
If the pointer "object" is a NULL-pointer then a var_object is created with the name that is present in the header-file. This is the name of the var_object when it was written to disk. When the object already exists then the contents of the old var_object will be lost.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

var_object  write_var_object

## *readfile*

***NAME***

readfile - read an image from file

***SYNOPSIS***

```
#include "im_proto.h"

IMAGE *readfile(char *filename, IMAGE *image, int xpos, int ypos)
```

***DESCRIPTION***

Read the image stored in file "filename" and put it in image "image". If "USE_NAME" (a NULL pointer) is specified as the image, a new image is created at position "xpos", "ypos", with the same name as the file. If an image is already present with that name, that image will be used. Several file formats are supported (see below), each of which have an obligatory extension. If a filename is supplied with no extension the function will append the obligatory extensions one at the time to find the file.

| | |
|---|---|
| ICS format | Two files per image are present, the data-file with the extension ".ids" and the header-file with the extension ".ics" |
| TIFF format | The read function is capable of reading TIFF-files according to the TIFF 6.0 specifications.The file must have an extension that starts with ".tif". The extensions used for finding a TIFF file are ".tif" and ".tiff". |
| JPEG format | The file must have the ".jpg" or ".jpeg" extension. |
| TCL format | The file must have the ".dat" extension. |
| AIM format | The data-files of the AIM format must have the ".im" extension. Data-files for which no header file with the extension ".hd" is present, are assumed to contain a 256 * 256 grey value image. |

***RETURN VALUES***

The pointer to the image in which the data was put, either an existing image or a newly created one.
NULL pointer on failure

***SEE ALSO***

ics_readfile  tiff_readfile  tcl_readfile  aim_readfile  jpeg_readfile  writefile

### *real_im*

***NAME***

real_im - get the real part of a complex image

***SYNOPSIS***

```
#include "im_proto.h"

int real_im(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

Take the real part of each element of the image "in" (a complex image) and store the results in the image "out". If "out" is a complex image then the result will be stored in the real part of each element of "out" and the imaginary part will be cleared. If "out" is not a complex image the result will be a float image "out".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

imaginary_im  complex_im

---

### *real_time_recognizer*

***NAME***

real_time_recognizer - real time recognizer

***SYNOPSIS***

```
#include "im_proto.h"

int real_time_recognizer(IMAGE *in, IMAGE *out, IMAGE *se, int thr,
int bound)
```

***DESCRIPTION***

The real time recognizer is a special implementation of t_morphology() for weighted structuring functions whose weights are restricted to -1, 0 and +1. For a description of how to use threshold morphology we refer to the documentation of t_morphology().

Please note that the name is a bit misleading. DO NOT EXPECT REAL TIME PERFORMANCE.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

arbit_erosion  arbit_dilation  hit_or_miss  t_morphology

---

## *reduce*

***NAME***

reduce - image reduce

***SYNOPSIS***

```
#include "im_proto.h"

int reduce(IMAGE *in, IMAGE *out, int hfact, int vfact, int dfact,
int adjust)
```

***DESCRIPTION***

Reduce image "in" with a horizontal factor "hfact", a vertical factor "vfact" and a depth factor "dfact" (3d only) by resampling pixels and store the result in image "out". If "adjust" is true (not zero) then the sizes of the image "out" will be set to fit the result. The dimensions of the output image are not adjusted by default due to various reasons.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

blow  fblow

## *remark*

***NAME***

remark - on-line remark facility for the user

***PLATFORM***

UNIX.

***SYNOPSIS***

```
void remark()
```

***DESCRIPTION***

If during a session of the package a bug or another problem occurs, the command remark can be issued to generate on-line a performance report. The command remark will prompt with a > sign. Following this prompt a line of text can be entered, terminated with a <newline> character. The command will prompt for more lines until an empty line is entered (only a <newline> character), which terminates a remark session. When the remark session is terminated the command will prompt for your name. The report will be stored into the remark file in the application standard directory.  This file is maintained by the manager of the package, who can  take action to solve the problem.

Restriction(s):

The text lines to be entered with this command should not exceed 80 characters (including the <newline> character).

***NOTE***

The SCIL remarkfile is located at the position indicated by the system environment variable SCIL_REMARKS, this position is determined at the start of the package. This file has to be writable for every user.

## *remove*

## *rename*

***NAME***

remove - remove a file

rename - rename a file

***SYNOPSIS***

```
int remove(char *filename)

int rename(char *oldname, char *newname)
```

***DESCRIPTION***

This function is an interface to the standard C function as implemented on the current system. The functionality of this function is:

remove() removes the named file, so that a subsequent attempt to open it will fail.

rename() changes the name of a file from "oldname" to "newname".

***RETURN VALUES***

non-zero if the attempt on the file fails

---

## *remove_holes*

***NAME***

remove_holes - remove small holes in image

***SYNOPSIS***

```
#include "im_proto.h"

int remove_holes(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

Removes single background pixels and sets of pixels which are not 4-connected to other background pixels within foreground objects in the image "in" and stores the result in the image "out".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

---

## *resample_perp*

***NAME***

resample_perp - resample perpendicular to curve

***SYNOPSIS***

```
#include "grey_2dp.h"

int resample_perp(IMAGE *input, VAR_OBJECT *xy, IMAGE *out, int
width, int fitlength, VAR_OBJECT *data, int threshold)
```

***DESCRIPTION***

Resample the image "input" perpendicular to the curve specified by the coordinate list "xy". For all N coordinate pairs in "xy" ("xy" should be a 2-dimensional VAR_OBJECT of type SHORT_T, with size 2 as the first dimension and size N as the second), a straight line is fitted through "fitlength" points, "fitlength"/2 points forward and "fitlength"/2 points backward in the list. The input image "input" is resampled on a line perpendicular to the fitted line segment and passing through its center of gravity. The resampling is done symmetrically with respect to the center point. The sample interval is one pixel. The new pixel values are calculated by linear interpolation between the nearest pixels in the original image.

The number of samples on the line is specified by "width", to which the width (first dimension) of the image "out" will be adjusted. For each sampling line, the data are stored into one row of "out".

Three parameters defining the sampling line segment are stored into the corresponding row of the floating point VAR_OBJECT "data", if this is specified, viz.: the x-coordinate of the center point into element 0 of the row, the Y coordinate into element 1 and the direction of the line (in radians) into element 2.

If the sampling exceeds the image boundaries, the corresponding entries in "out" are set to -1.

With the parameter "threshold" it is possible to specify the area which cannot be used for the computations. Pixels with a value less than or equal to this parameter are considered as prohibited.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

maximum_trace maximum_cost_path back_project drawcurve

## *retrieve_object_list*

***NAME***

retrieve_object_list - retrieve last labeled object list

***SYNOPSIS***

```
#include "im_aio.h"

LIST *retrieve_object_list(void)
```

***DESCRIPTION***

Retrieves a pointer to the most recently labeled object list. Since it is the users responsibility to free object lists it is important to retrieve a pointer to the most recently labeled list. If the user forgets to save the list returned by list_label(), retrieve_object_list() can give some relief.

***NOTE***

retrieve_object_list() is part of the AIO package

***RETURN VALUES***

A pointer to the list with the last labeled objects

***SEE ALSO***

measure

---

## *RGB_gamma_correction*

***NAME***

RGB_gamma_correction - gamma correction on a RGB color-image

***SYNOPSIS***

```
#include "color2dp.h"

int RGB_gamma_correction(IMAGE *in, IMAGE *out, double r_gamma,
double g_gamma, double b_gamma)
```

***DESCRIPTION***

RGB_gamma_correction() performs a gamma correction on the R-, G- and B-channel of the RGB color-image "in" and store the result in the image "out". The gamma-value for each of the channels can be specified separately by the parameters "r_gamma" for the R-channel, "g_gamma", for the G-channel and "b_gamma" for the B-channel.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

---

## *rhull*

***NAME***

rhull - restricted convex hull detection

***SYNOPSIS***

```
#include "im_proto.h"

int rhull(IMAGE *in, IMAGE *out, int dist)
```

***DESCRIPTION***

Calculate a restricted convex hull of each object in the labeled image "in" and store the result in image "out". For each object in "in", all combinations of two contour points with Euclidean distance less than or equal to "dist", are connected by a straight line. If a background pixel is found on such a line, it is added to the original object. This operation closes all holes in an object which are less than "dist" wide. The contour of an object is also smoothed, gaps with a length less than "dist" are completely filled.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

hull  label  objectsize  small_object_removal

---

## *rm_list*

***NAME***

rm_list - remove list with objects

***SYNOPSIS***

```
#include "im_aio.h"

LIST *rm_list(LIST *list)
```

***DESCRIPTION***

| | | |
|---|---|---|
| list | - | List with objects |

rm_list() removes the list with objects pointed to by "list" and returns allocated space to the system.

***RETURN VALUES***

NULL

***SEE ALSO***

list_label  measure  retrieve_object_list

---

## *rm_object*

***NAME***

rm_object - mark object to be removed from the list

***SYNOPSIS***

```
#include "im_aio.h"

void rm_object(LIST *link)
```

***DESCRIPTION***

| | | |
|---|---|---|
| link | - | Link pointing to object |

rm_object() marks the object pointed to by "link", to be removed from the list on the next call to the update() function.

***NOTE***

The object is not removed from the image. You need to call hide_object() to remove an object from an image.

***EXAMPLE***

```
To remove objects touching the edge of an image:

#include "image.h"
#include "im_aio.h"
LIST *l, *o;

readfile("cermet",a,0,0);
threshold(a,b,128);
invert_im(b,b);
l = list_label(b,c,8,0);
FORALL(o,l) if(edge_object(c,o)) rm_object(o);
l = update(l);
/*
* Now to prove that the objects are no longer in the list
*/
FORALL(o,l) copy_object(c,d,o);
l = rm_list(l);
```

***RETURN VALUES***

None

***SEE ALSO***

hide_object  update

## *rm_silo*

***NAME***

rm_silo - destroys an image-silo

***SYNOPSIS***

```
#include "silo.h"

int rm_silo(SILOPTR siloptr)
```

***DESCRIPTION***

| | | |
|---|---|---|
| siloptr | - | Pointer to the image-silo. |

A routine to remove a silo from the filing system. Returns all allocated space to the system.

***NOTE***

It is also legal to remove the file. This routine is meant to be used inside a program.

***RETURN VALUES***

IE_OK (1) on succes
Negative error status on failure (see im_error.h)

---

## *rm_silo_object*

***NAME***

rm_silo_object - remove an image from image-silo

***SYNOPSIS***

```
#include "silo.h"

int rm_silo_object(SILOPTR siloptr, int silo_key)
```

***DESCRIPTION***

| | | |
|---|---|---|
| siloptr | - | Pointer to the image silo |
| silo_key | - | Key entry of the image in the silo |

Empties the entry at position "silo_key" in the silo "siloptr". It also adjusts the internal entry list.

***RETURN VALUES***

Always IE_OK (1)

---

## *rmvar*

***NAME***

rmvar - clear interpreter

***SYNOPSIS***

```
rmvar
```

***DESCRIPTION***

Clears the interpreter hereby removing old programs, variables, structure descriptions typedefs and preprocessor defines.

---

## *roberts_diff*

***NAME***

roberts_diff - Roberts gradient edge operator

***SYNOPSIS***

```
#include "im_proto.h"

int roberts_diff(IMAGE *in, IMAGE *out, int fsize, int mode)
```

***DESCRIPTION***

Differential edge detection based upon the Roberts gradient, an estimation of the local gradient. Within the moving window in image "in", with dimensions "fsize" * "fsize" the differences between the pixel values at the end points of the diagonals of the window are calculated. The output value, which is stored into image "out", is calculated from these differences according the "mode" specified:

| | |
|---|---|
| sqrt (1) | the output value is the square root of the sum of the quadratic differences |
| sum (0) | the output value is the sum of the absolute values of the differences |

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

prewitt_diff  sobel_diff  laplace  prewitt_temp  kirsch_temp  robinson_temp

---

## *robg*

***NAME***

robg

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See roberts_diff

## ***robinson_temp***

***NAME***

robinson_temp - edge detection filter

***SYNOPSIS***

```
#include "im_proto.h"

int robinson_temp(IMAGE *in, IMAGE *out, IMAGE *direction, int flag)
```

***DESCRIPTION***

Template type edge detection based upon the Robinson operator. Within the moving window in the image "in", with dimensions 3 * 3, eight convolutions with the following masks are calculated:

|  | (0) |  |  | (1) |  |  | (2) |  |  | (3) |  |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 0 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 0 |
| -2 | 0 | 2 | -1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | -1 |
| -1 | 0 | 1 | -2 | -1 | 0 | -1 | -2 | -1 | 0 | -1 | -2 |

|  | (4) |  |  | (5) |  |  | (6) |  |  | (7) |  |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | -1 | 0 | -1 | -2 | -1 | -2 | -1 | -2 | -1 | 0 |
| 2 | 0 | -2 | 1 | 0 | -1 | 0 | 0 | 0 | -1 | 0 | 1 |
| 1 | 0 | -1 | 2 | 1 | 0 | 1 | 2 | 1 | 0 | 1 | 2 |

The output value is the maximum of the results of all these convolutions. It is stored into "out", in the pixel corresponding with the central pixel of the window. The sequence number of the convolution mask with the maximum result is an estimate of the direction of the first derivative and it is stored in the image "direction", if this is specified ("flag"=1).

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

laplace prewitt_temp kirsch_temp prewitt_diff roberts_diff sobel_diff

## *roi_define*

***NAME***

roi_define - define a region of interest

***SYNOPSIS***

```
#include "im_infra.h"

IMAGE *roi_define(char *name, IMAGE *parent, int sx, int sy, int sz,
int width, int height, int depth, BOOL_MASK *mask)
```

***DESCRIPTION***

roi_define() creates a region of interest with name "name" in the parent image "parent" at ("sx","sy","sz") with dimensions "width"*"height"*"depth". The last parameter is an optional Boolean mask (can also be NULL) which allows for a roi definition with an arbitrary shape. A roi has the same status as any other image except for the fact that a roi can not be defined inside another roi.

Not only rectangular shaped regions of interest are possible in SCIL_Image, but also arbitrary shaped ones. To create one, a binary image has to be converted into a Boolean mask by use of the function get_bool_mask(). The pointer to this Boolean mask must then be specified as the last parameter of roi_define().

***NOTE***

The normal behavior of output images in Image is that they automatically are adjusted to the correct size and type suited for the result of the operation. ROIs however only allow type changes but have fixed dimensions. The only way to change the dimensions of a roi is to use change_image_size.

***RETURN VALUES***

A pointer to the roi on success.
NULL pointer on failure to create the roi.

***SEE ALSO***

destroy_image  get_bool_mask  change_image_size

## *rotate*

***NAME***

rotate - image rotation

***SYNOPSIS***

```
#include "im_proto.h"

int rotate(IMAGE *in, IMAGE *out, int iter)
```

***DESCRIPTION***

Rotate image "in" over "iter"*90 degrees and store the result in image "out". If "iter" is positive the operation is performed clockwise, if "iter" is negative counterclockwise.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

mirror

---

***NAME***

```
run - execute the program in the program buffer
```

***SYNOPSIS***

run

***DESCRIPTION***

The run command is used to execute a previously loaded file. All old variables are deleted.

---

## *S_Append*

## *S_BreakList*

## *S_CloseList*

## *S_CopyList*

## *S_Delete*

## *S_FindItem*

## *S_FreeList*

## *S_Insert*

## *S_Length*

## *S_Prefix*

## *S_SortList*

***NAME***

S_Append, S_BreakList, S_CloseList, S_CopyList, S_Delete, S_FindItem, S_FreeList, S_Insert, S_Length, S_Prefix, S_SortList - linked list functions

***SYNOPSIS***

```
#include "linklist.h"

LIST *S_Append(INFO *item, LIST *list)

LIST *S_Insert(INFO *item, LIST *list)

LIST *S_Prefix(INFO *item, LIST *list)

LIST *S_Delete(LIST *item, FREEFUNC FreeInfo)

int S_FreeList(LIST *list, FREEFUNC FreeInfo)

int S_Length(LIST *list)

LIST *S_FindItem(INFO *item, LIST *list)

LIST *S_SortList(LIST *list, COMPAREFUNC compare)

LIST *S_BreakList(LIST *list)

LIST *S_CloseList(LIST *list)

LIST *S_CopyList(LIST *list)
```

***DESCRIPTION***

These functions are used for generic linked list creation and maintenance. A linked list is build using the following structure:

```
typedef struct List {
      struct List *next;
      struct List *prev;
      INFO        *info;
} LIST;
```

A linked list is a number of LIST structures linked together by their "next" and "prev" fields to form a double linked list. Any information can be stored in the list as long as a pointer to the data is available. The pointer is stored in the "info" field (as a void pointer). The number of lists and the number of elements in each list is not limited by these functions in any way.

S_Append(), S_Insert() and S_Prefix() are used to build a list. They allocate a new LIST structure, store the "item" pointer in the info field and insert the LIST structure at the correct position in the list "list" taking care that the "next" and "prev" fields of their neighbors (if present) are updated correctly. S_Append() appends the new element to the end of linked list "list". S_Prefix() inserts the new element at the start of list "list". S_Insert() inserts the new element just before the element pointed to by "list". If "list" is NULL a new linked list is started and the returned LIST pointer should be stored for future reference of the list.

S_Delete() and S_FreeList() are used to remove certain elements from the list or the entire list respectively. S_Delete() deletes element from linked list. The function "freefunc" is used to destroy the element attached to the list. When the element was allocated using malloc(), the function free() can be used. Complicated elements e.g. structures with pointere to other structures should be destroyed by dedicated destroy-functions. NULL is allowed if the element is not to be destroyed.

S_Length() returns the length of the list i.e. the number of elements in the list.

S_FindItem() searches in the list "list" for the element "item" and returns a pointer to the LIST structure that contains the element. If it cannot find "item", NULL is returned

S_CloseList() and S_BreakList() convert a NULL terminated list in a circular list and vice versa. When building a linked list, the "prev" field the first item and the "next" field of the last element are NULL terminated. S_CloseList() connects these those elements creating a circular list. S_BreakList() breaks the circular open again at the LIST structure pointed to by "list" making it the new start of the list.

S_SortList() sorts the list according to the supplied comparison function "compare". The list is sorted using the qsort() function.

S_CopyList() make a copy of a list. This second list contains physically the same elements as the first list i.e. the elements in the list are not duplicated. Removing an element from one list will leave a pointer dangling it the other list.

***RETURN VALUES***

S_Append(), S_Prefix(), S_Insert() and S_CopyList() all return NULL if there is not enough memory to create the new LIST structure(s). If successful, S_Append() and S_Prefix() return a pointer to the start of the (changed) list, S_Insert() returns a pointer to the new member of the list. S_CopyList() returns a pointer to the start of the new list.

S_Delete() returns a pointer to the "next" member of the list, if the deleted member was the last of the list, a pointer to the previous member is return. In case the list contained only one member, or "list" was NULL, NULL is returned. S_FreeList() returns FALSE (0) is list was NULL, otherwise it returns TRUE (1).

S_Length() returns the length of the list. S_FindItem() returns a pointer to a LIST struct or NULL if not found. S_BreakList() and S_CloseList() return their argument. S_SortList() returns a LIST pointer to the sorted list or NULL if the list was empty or if it cannot allocate memory to store the sorted list.

---

## *saxis*

***NAME***

saxis - obtain short axis of the fitted ellipse of an object

***SYNOPSIS***

```
#include "im_aio.h"

double saxis(LIST *link)
```

***DESCRIPTION***

| | | |
|---|---|---|
| link | - | Link pointing to object |

AIO primitive to obtain value of an object feature

saxis() returns the length of the short axis of the fitted ellipse of the object pointed to by "link" if this has previously been measured.

***RETURN VALUES***

length of the short axis of the fitted ellipse of object
0.0 if link is not an object or if short axis has not been measured

***SEE ALSO***

measure  object_shape_meas  object_dens_meas  laxis

---

## *scanf*

## *fscanf*

## *sscanf*

***NAME***

scanf, fscanf, sscanf - formatted input conversion

***SYNOPSIS***

```
#include <stdio.h>

int scanf(char *format, ...)

int fscanf(FILE *stream, char *format, ...)

int sscanf(char *s, char *format, ...)
```

***DESCRIPTION***

These functions are interface functions to the standard C functions as implemented on the current operating system. The functionality of these functions is:

scanf() reads from the standard input stream stdin. fscanf() reads from the named input "stream". sscanf() reads from the character string "s". Each function reads characters, interprets them according to a format, and stores the results in its arguments. Each expects as arguments a control string "format", described below, and a set of pointer arguments indicating where the converted input should be stored.

The control string usually contains conversion specifications, which are used to direct interpretation of input sequences. The control string may contain:

- Blanks, tabs or newlines, which match optional white space in the input.

- An ordinary character (not %) which must match the next character of the input stream.

- Conversion specifications, consisting of the character %, an optional assignment suppressing character *, an optional numerical maximum field width, and a conversion character.

A conversion specification directs the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument, unless assignment suppression was indicated by *. An input field is defined as a string of non-space characters; it extends to the next inappropriate character or until the field width, if specified, is exhausted.

The conversion character indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. The following conversion characters are legal:

| | |
|---|---|
| % | a single ’%’ is expected in the input at this point; no assignment is done. |

d a decimal integer is expected; the corresponding argument should be an integer pointer.

o an octal integer is expected; the corresponding argument should be a integer pointer.

x a hexadecimal integer is expected; the corresponding argument should be an integer pointer.

s a character string is expected; the corresponding argument should be a character pointer pointing to an array of characters large enough to accept the string and a terminating ’\0’, which will be added. The input field is terminated by a space character or a newline.

c a character is expected; the corresponding argument should be a character pointer. The normal skip over space characters is suppressed in this case; to read the next non-space character, try "%1s". If a field width is given, the corresponding argument should refer to a character array, and the indicated number of characters is read.

f a floating point number is expected; the next field is converted accordingly and stored through the corresponding argument, which should be a pointer to a float. The input format for floating point numbers is an optionally signed string of digits possibly containing a decimal point, followed by an optional exponent field consisting of an E or e followed by an optionally signed integer.

[ indicates a string not to be delimited by space characters. The left bracket is followed by a set of characters and a right bracket; the characters between the brackets define a set of characters making up the string. If the first character is not circumflex (^), the input field is all characters until the first character not in the set between the brackets; if the first character after the left bracket is ^, the input field is all characters until the first character which is in the remaining set of characters between the brackets. The corresponding argument must point to a character array.

The conversion characters d, o and x may be capitalized or preceded by l to indicate that a pointer to long rather than to int is in the argument list. Similarly, the conversion characters e or f may be capitalized or preceded by l to indicate a pointer to double rather than to float. The

conversion characters d, o and x may be preceded by h to indicate a pointer to short rather than to int.

The scanf functions return the number of successfully matched and assigned input items. This can be used to decide how many input items were found. The constant EOF is returned upon end of input; note that this is different from 0, which means that no conversion was done; if conversion was intended, it was frustrated by an inappropriate character in the input.

***EXAMPLE***

For example, the call

```
int i; float x; char name[50];
scanf("%d%f%s", &i, &x, name);
```

with the input line

```
25 54.32E-1 thompson
```

will assign to i the value 25, x the value 5.432, and name will contain "thompson\0".

Or,

```
int i; float x; char name[50];
scanf("%2d%f%*d%[1234567890]", &i, &x, name);
```

with input

```
56789 0123 56a72
```

will assign 56 to i, 789.0 to x, skip "0123", and place the string "56\0" in name. The next call to getchar will return ’a’.

***RETURN VALUES***

The scanf functions return EOF on end of input, and a short count for missing or illegal data items.

***SEE ALSO***

atof  getc  printf

## *searchfile*

## *rel_searchfile*

***NAME***

searchfile, rel_searchfile - find and open a file

***SYNOPSIS***

```
#include "support.h"

FILE *searchfile(char *name, char *envvar, char *pathret)

FILE *rel_searchfile(char *name, char *envvar, char *pathret)
```

***DESCRIPTION***

searchfile(), searches for the file whose name is given in "name". If it can not open the file using "name", it will remove any directory names from "name" and will try to open the file in the current directory. If still unsuccessful, it will search for the file in all the directories that are listed in the environment variable specified in "envvar". When the file is found, it is opened using fopen() with read-only permission ("r"). The entire path-name of the file is stored in the supplied buffer "pathret", that is assumed to be at least 256 bytes long. The FILE pointer to the still open file is returned.

rel_searchfile() searches and opens files in almost the identical way as searchfile(). The only difference is that any preceding directory names are not stripped from "name". It appends the entire path-name in "name" to the directory names in "envvar".

The various directories specified in the environment variable "envvar" must be separated by a separator. On UNIX systems, this is a ":" (colon), on Ms-Windows and Macintosh systems it is the ";" (semicolon). Additionally it is allowed on MS-Windows systems to use the "/" (forward slash) as a directory separator in path-names, this to reduce the chance of errors when using the "\" (backslash) which in C-strings must be escaped with another backslash.

***NOTE***

(rel_)searchfile() uses fopen() with type "r" to open the files. This means that when using searchfile() to open files that contain binary data (non-text files) on some platform translation of carriage-return characters will occur. For binary files on MS-Windows and Macintosh this means that you must close the file using fclose() and reopen it with fopen( pathret, "rb"). Remember to use the path-name from "pathret" to reopen the file, the correct path-name for the file you just closed is in that buffer.

***RETURN VALUES***

searchfile() and rel_searchfile() return the FILE pointer of the opened file which was searched for or NULL if the file cannot be found.

## *set_aio_disp*

***NAME***

set_aio_disp - enable/disable immediate display of AIO objects

***SYNOPSIS***

```
int set_aio_disp(int mode)
```

***DESCRIPTION***

When "mode" is 1 then the objects in labeled images are displayed at once when they are copied or hidden using the functions hide_object(), copy_object(), g_copy_object() and g_copy_object_to(). The objects will be displayed without having to display the entire image.

When "mode" is 0, changes will only become visible when the image is (re)displayed. This is useful when copying or hiding a number of objects.

***RETURN VALUES***

status of the AIO display mode (value of "mode" on the last call to set_aio_disp())

***SEE ALSO***

hide_object  copy_object  g_copy_object  g_copy_object_to

---

## *set_border*

***NAME***

set_border - set the border of an image

***SYNOPSIS***

```
#include "im_proto.h"

int set_border(IMAGE *out, double value, int top, int right, int bot,
int left, int z_min, int z_max)
```

***DESCRIPTION***

The border of image "out" is given a value "value". The size (=thickness) of the border for each side of the image is given by "up", "right", "bot", "left", "z_min", "z_max"

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***NOTE***

The parameters "z_min" and "z_max" are ignored for 2D images.

***SEE ALSO***

set_int

---

## *set_clut*

***NAME***

set_clut - attach a color lookup table to an image

***SYNOPSIS***

```
#include "im_infra.h"

int set_clut(IMAGE *image, CLUT *clut, int disp)
```

***DESCRIPTION***

Attaches the color lookup table "clut" to "image". If the flag "disp" is set (=1) then the image will be (re)displayed to show the effect.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

create_clut

---

## *set_color*

***NAME***

set_color - set pixel value of color image

***SYNOPSIS***

```
#include "color2dp.h"

int set_color(IMAGE *image, int red_val, int green_val, int blue_val)
```

***DESCRIPTION***

set_color() sets all pixels of "image" to the RGB value given by "red_val", "green_val", blue_val".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

clear_im  make_color_im

---

## *set_common_line*

***NAME***

set_common_line - fill the common_line structure in one pass

***SYNOPSIS***

```
#include "im_infra.h"

void set_common_line(COMMON_LINE *com_line, int type, void *data, int
x, int y, int z, int t, int chan, double h_min, double h_max)
```

***DESCRIPTION***

This function fills the COMMON_LINE structure "com_line" in one pass with the values specified as the arguments. It is used in the low-level routines of the convert() operation. The function is meant for programmers convenience.

"com_line" is pointer to the COMMON_LINE structure that must be filled.

"type" specifies which type of data is used, COM_LONG or COM_DOUBLE.

"data" is a pointer to the allocated memory to store the data of one image line.

"x", "y", "z", "t" and "chan" are the sizes and the position of the image line in the memory pointed at by "data"

"h_min" and "h_max" are the minimum and maximum value of the data in the entire source image of the convert operation.

***RETURN VALUES***

None

***SEE ALSO***

convert

## *set_comp_margin*

***NAME***

set_comp_margin - set space between images in composite photo

***SYNOPSIS***

```
#include "silo.h"

void set_comp_margin(int size)
```

***DESCRIPTION***

| | | |
|---|---|---|
| size | - | Size of margin |

Function to define the space between sub-images in a composite photo.

***RETURN VALUES***

None

***SEE ALSO***

start_comp  silo_to_comp

---

## *set_complex*

***NAME***

set_complex - set pixel value of image

***SYNOPSIS***

```
#include "im_proto.h"

int set_complex(IMAGE *im, double real_part, double imaginary_part)
```

***DESCRIPTION***

Set all pixels in image "im" to the complex value given by "real_part" and "imaginary_part".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

clear_im

---

## *set_dialog_pos*

***NAME***

set_dialog_pos - set the position of the dialog box

***PLATFORM***

Unix, Macintosh.

***SYNOPSIS***

```
#include "md_gen.h"

int set_dialog_pos(int x, int y)
```

***DESCRIPTION***

set_dialog_pos() sets the position of the dialog box to ("x","y").

***RETURN VALUES***

None

***SEE ALSO***

set_menu_pos

## *set_display_mode*

***NAME***

set_display_mode - set the display mode for an image

***PLATFORM***

UNIX, Macintosh.

***SYNOPSIS***

```
#include "disp_p.h"
#include "dmodes.h"

int set_display_mode(IMAGE *image, int mode, int global, int
direct_display)
```

***DESCRIPTION***

set_display_mode() sets the scaling mode used for displaying the image "image". The "mode" must be set for each image separately. Only one mode can be set simultaneously for an image. When specifying more than one mode, only one mode is set, which one is not defined. The currently implemented modes are defined in the include file "dmodes.h", they are:

| | |
|---|---|
| DM_NORMAL | Normal display. |
| DM_LIN_STRETCH | Linear stretched display. |
| DM_LOG_STRETCH | Logarithmic stretched display. |
| DM_LIN_ERROR | Linear stretched error display. |
| DM_LOG_ERROR | Logarithmic stretched error display. |
| DM_SIGMOID | Sigmoidal stretched display. |

In an image with display mode DM_LIN_ERROR or DM_LOG_ERROR the value 0 is displayed as greyvalue 127. Positive errors are displayed between 127 and 255. Negative errors are displayed between 0 and 127.

"global" is in effect only for 3D images:

| | |
|---|---|
| global = No (0): | The minimum and maximum value used for stretching are determined only from the current slice. |
| global = Yes (1): | The min an max value used for stretching are determined from the entire image |

"direct_display" indicates if the image should be (re)displayed directly Yes(1) or not No(0).

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

set_sigmoid_shape  set_dither_mode  get_display_mode  show_dmode_flags

## set_display_slice

***NAME***

set_display_slice - display a slice of a image

***SYNOPSIS***

```
#include "disp_p.h"

int set_display_slice(IMAGE *image, int slice)
```

***DESCRIPTION***

Displays slice number "slice" of image "image". The slice number that is being displayed at the moment is the field "slice" of the IMAGE structure.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

display_image  next_plane

## *set_dither_mode*

***NAME***

set_dither_mode - perfect display of images

***PLATFORM***

UNIX, Macintosh.

***SYNOPSIS***

```
#include "disp_p.h"

int set_dither_mode(IMAGE *image, int mode, int direct_display)
```

***DESCRIPTION***

Images that have a color lookup-table attached are normally displayed by mapping the entries of the table the color the is the nearest in the SCIL_Image color-lookup table. Due to the limited number of colors that will fit in a color-table at one time the effect can be very poor. By turning on color-dithering (for each image separately) the display routine will show the image in "true color" (a near perfect approximation).

set_dither_mode() turns on the color-dithering for the image "image" when "mode" is set to 1. It is turned off when "mode" is set to 0. "direct_display" indicates that the image should be (re)displayed directly Yes(1) or not No(0).

***NOTE***

Turning this option on shows a significant performance penalty for the display update times

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

set_display_mode

## *set_float*

***NAME***

set_float - set pixel value of an image

***SYNOPSIS***

```
#include "im_proto.h"

int set_float(IMAGE *im, double constant)
```

***DESCRIPTION***

Set all pixels in image "im" to the floating point value "constant".

***NOTE***

For more powerful image arithmetic expressions (scaling, adding offsets, etc.), use the function eval() .

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

clear_im  eval

---

## *set_im_type*

***NAME***

set_im_type - change the type of an image

***SYNOPSIS***

```
#include "im_infra.h"

int set_im_type(IMAGE *im, int type)
```

***DESCRIPTION***

The type of image "im" is changed to "type". The data of the image is lost.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

---

## *set_image_flag*

## *clear_image_flag*

***NAME***

set_image_flag, clear_image_flag - set/clear the image flags.

***SYNOPSIS***

```
#include "im_proto.h"

void set_image_flag(IMAGE *im, int flag)

void clear_image_flag(IMAGE *im, int flag)
```

***DESCRIPTION***

An image contains several flags that can be used to indicate special treatment of such an image flag. These flags can be set with set_image_flag() and cleared with clear_image_flag(). After a flag has been set/cleared, the image publishes a SPB_NEWSTATE message. Currently the following flags are defined:

| | |
|---|---|
| READ_ONLY | (bit 0, integer value 1), image is read-only, the infrastructure does not allow you to use the image as an output image. The image size and/or type cannot be altered. |
| NOT_IN_DIALOG | (bit 1, integer value 2), signals to the GUI that the image should not be shown in dialog boxes. |
| NO_AUTO_POINT | (bit 2, integer value 4), signals to the GUI that when pointing in the image viewer with the mouse, the pixel information should not be shown. |
| NO_AUTO_DISPLAY | (bit 3, integer value 8), signals to the GUI that the image should not automatically be displayed on changes to the image. |

***RETURN VALUES***

None

## *set_image_interaction*

## *handle_events*

***NAME***

set_image_interaction - disable SCIL_Image event loop

handle_events - call the SCIL_Image event loop once

***SYNOPSIS***

```
#include "im2scil.h"

int set_image_interaction(int mode)

void handle_events(void)
```

***DESCRIPTION***

When running an interpreted program, the event loop of SCIL_Image is called after each statement. This can cause event driven programs to miss out on events. set_image_interaction(), when called with "mode" = On (1), prevents the interpreter from calling the event loop after each statement.

When set_image_interaction() has been called with "mode" = On from within an interpreted program, handle_events() should be called in its event loop to allow SCIL_Image to update images, operate the menu etc.

A program that uses set_image_interaction should also call it with "mode" is Off (0) when it exits, to restore the interpreter to its prior state.

***RETURN VALUES***

None

***SEE ALSO***

handle_events  poll_mouse  point_im  im_input_func  del_im_input_func im_exposure_func  del_im_exposure_func

## *set_int*

***NAME***

set_int - set pixel value of an image

***SYNOPSIS***

```
#include "im_proto.h"

int set_int(IMAGE *im, int constant)
```

***DESCRIPTION***

Set all pixels in the image "im" to the integer value "constant"

***NOTE***

For more powerful image arithmetic expressions (scaling, adding offsets, etc.), use the function eval() .

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

clear_im  eval

## *set_line_editor*

***NAME***

set_line_editor - switch the line editor mode

***PLATFORM***

UNIX.

***SYNOPSIS***

```
void set_line_editor(int mode)
```

***DESCRIPTION***

This function can be used to change the operation mode of the command line editor. By default the command line editor emulates the "vi" editor. The "vi" mode can also be set by using "mode" = Vi (1). The "vi" mode is described in the main manual of SCIL_Image. The  other operating mode of the command line editor is controlled by the cursor keys and some other <Control> keys as described below. This mode is enabled by specifying "mode" = Cursor (0). The keys used in that mode are the following:

| | |
|---|---|
| ^L | Print the history list. |
| ^A,^P,Up | Walk back through the history |
| ^N,Down | Walk forward through the history |
| ^F,Right | Jump to the next character in the current line. |
| ^B,Left | Jump to the previous character in the current line. |
| ^R | Reprint the current line. |
| ^E | Jump to the end of the current line. |
| ^K | Clear the rest of the current input line. |
| ^X,^U | Discard the complete current line. |
| ^H,DEL | Delete the character before the current cursor position. |
| ^D (EOF) | Discard the complete current line. If EOF is typed twice the line "exit(0);" will be passed to the command decoder. |

***RETURN VALUES***

None

## *set_menu_pos*

***NAME***

set_menu_pos - set the position of the menu panel

***PLATFORM***

Unix.

***SYNOPSIS***

```
#include "md_gen.h"

int set_menu_pos(int x, int y)
```

***DESCRIPTION***

set_menu_pos() sets the position of the control_panel to ("x","y").

***RETURN VALUES***

None

***SEE ALSO***

set_dialog_pos

## *set_rgb_bits*

***NAME***

set_rgb_bits - set the bitplane colors for MULTI_LUT_T lookup table

***SYNOPSIS***

```
#include "im_infra.h"

int set_rgb_bits(int r_bit, int g_bit, int b_bit)
```

***DESCRIPTION***

With this function the bitplanes that will be displayed in color when using a MULTI_LUT_T lookup table can be specified. Bitplane "r_bit" will be displayed in red, "g_bit" in green and "b_bit" in blue.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

create_clut

## *set_RGB2XYZ_matrix*

## *set_RGB2XYZ_mvalues*

## *print_RGB_matrices*

***NAME***

set_RGB2XYZ_matrix - set the RGB to XYZ conversion matrix

set_RGB2XYZ_mvalues - set the RGB to XYZ conversion matrix values

print_RGB_matrices - display the RGB to XYZ conversion matrix values

***SYNOPSIS***

```
#include "color2dp.h"

void set_RGB2XYZ_matrix(int matrix_type)

void set_RGB2XYZ_mvalues(double m11, double m12, double m13, double
m21, double m22, double m23, double m31, double m32, double m33)

void print_RGB_matrices(void)
```

***DESCRIPTION***

Conversion between the RGB and XYZ color-model is done by a matrix multiplication:

```
| X |   | m11 m12 m13 | | R |
| Y | = | m21 m22 m23 | | G |
| Z |   | m31 m32 m33 | | B |
```

The correct conversion matrix depends on many factors such as input device, display monitor etc. To influence the conversion, the conversion matrix can be changed. set_RGB2XYZ_matrix() sets the matrix to a small number of predefined values via "matrix_type" (listed below). set_RGB2XYZ_mvalues() can be used to set the matrix to any desired value via its parameters "m11" … "m33".

| Matrix type | id |
|---|---|
| NTSC_T | 1 |
| PAL_T | 2 |
| CIE_T | 3 |
| smpte_T | 4 |
| Rec709_T | 5 |

After the values have been set, immediately the inverse matrix is calculated which performs the XYZ to RGB conversion.

print_RGB_matrices() displays the values of both the conversion matrices, the RGB to XYZ matrix and the XYZ to RGB matrix.

***RETURN VALUES***

None

***SEE ALSO***

convert_cmodel

---

## *set_roi_clean_display*

***NAME***

set_roi_clean_display - redisplay parent of a ROI when displaying a ROI

***SYNOPSIS***

```
#include "im_infra.h"

void set_roi_clean_display(int mode)
```

***DESCRIPTION***

When a region of interest is the output for an operation its parent image will also be redisplayed depending upon the "mode" flag. This flag can be set using set_roi_clean_display().

Only when the image type of the ROI has changed as a result of the operation, the influence of the flag will become visible. In that case the parent image has become empty except for the ROI as result of the change of image type. When the flag is "1" the parent image will  be displayed again and only the ROI can be seen. If the flag has been set to "0" only the ROI will be displayed, leaving the rest of the  parent image visible on the screen. The parent image has become empty however. The flag can be set using "set_roi_clean_display". Its initial value is "1".

The example below shows that the ROI has become of another type and therefore the type of the parent image changed as well. The data of  the image however is still visible but no longer present as can be seen when the image is copied to another image.

***EXAMPLE***

```
readfile("trui", A, 0, 0);
roi_define("roi1", A, 64, 64, 0, 128, 128, 1, 0);
set_roi_clean_display(0);
threshold( roi1, roi1, 128);
copy_im( A, B);
```

***RETURN VALUES***

None

---

## *set_roi_pos*

## *set_roi_mask*

## *set_roi_parent*

***NAME***

set_roi_pos, set_roi_mask, set_roi_parent - roi manipulation functions

***SYNOPSIS***

```
#include "im_infra.h"

int set_roi_pos(IMAGE *roi_im, int sx, int sy, int sz)

int set_roi_mask(IMAGE *roi_im, BOOL_MASK *mask)

int set_roi_parent(IMAGE *roi_im, IMAGE *parent)
```

***DESCRIPTION***

These functions manipulate ROI-images created with roi_define(), their functionality can also be achieved by calling roi_define() with new parameters.

set_roi_pos() moves the origin of the ROI "roi_im" in the parent image to ("sx","sy","sz") without changing the sizes of the ROI. The ROI is only moved if it remains totally within the parent image.

set_roi_mask() gives the ROI "roi_im" a new Boolean "mask". Like with roi_define() the BOOL_MASK has to be created first using the function get_bool_mask().

set_roi_parent() changes the parent image of the ROI "roi_im" to "parent". As soon as the "roi_im" is used for input, its type will be changed to that of the new parent. If the RIO does not fit in the new parent image, the parent of the ROI is not changed.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

roi_define  get_bool_mask

## set_screen_gamma

***NAME***

set_screen_gamma - specify the gamma correction for your screen

***PLATFORM***

UNIX, MS-Windows.

***SYNOPSIS***

```
#include "disp_p.h"

int set_screen_gamma(double gamma)
```

***DESCRIPTION***

Gamma correction is used to correct for nonlinear responses on display devices (monitors). The value "gamma" is used to calculate a curve that describes the response of the device. This curve is used to correct for the nonlinear response.

If used, this function must be called prior to the call to initimage() or init_scil_image() in the scilinit file, if not called, a default of 1.0 will be assumed (no correction). Calling this function AFTER the display is initialized has no effect on the display of images.

***RETURN VALUES***

None

## *set_sigmoid_shape*

***NAME***

set_sigmoid_shape - determine the sigmoid shape used for the sigmoid display mode

***PLATFORM***

UNIX, Macintosh.

***SYNOPSIS***

```
#include "disp_p.h"

set_sigmoid_shape(float slope, float bending_point)
```

***DESCRIPTION***

This routine sets two global variables that describe a sigmoid used for creating an alternative lookup table for the display of images. "slope" determines the rate of ascent of the sigmoid. "bending point" determines the position of the bending point of the sigmoid.

***RETURN VALUES***

None.

***SEE ALSO***

set_display_mode  set_dither_mode

## *set_start_pos*

## *set_start_sizes*

***NAME***

set_start_pos - set the initial position of the four windows

set_start_sizes - set the initial size of the four windows

***SYNOPSIS***

```
#include "im2scil.h"

void set_start_pos(int x1, int y1, int x2, int y2, int x3, int y3,
int x4, int y4)

void set_start_sizes(int w1, int h1, int w2, int h2, int w3, int h3,
int w4, int h4)
```

***DESCRIPTION***

With set_start_pos() the initial position of the display windows of the images created by default_images() is set. This function must be called before default_images(). The first display window that default_images() creates gets position ("x1","y1"). The second is put at ("x2","y2"), etc.

set_start_sizes() sets the initial sizes of the display windows of the images created by default_images(). This especially useful on small displays. This function must be called before default_images. The first window that initimage puts on the display then gets sizes ("w1","h1"). The second is gets size ("w2","h2"), etc.

***RETURN VALUES***

None

***SEE ALSO***

default_images  set_window_pos  set_window_size  natural_window_size

## *set_tiff_compression*

***NAME***

set_tiff_compression - enable compression of TIFF data

***SYNOPSIS***

```
#include "im_proto.h"

void set_tiff_compression(int enable)
```

***DESCRIPTION***

The data in a TIFF file can either be compressed or uncompressed. When writing TIFF files, this function can be used to specify that the data should be compressed or not. If "enable" is 1, all subsequent writing to TIFF files will be compressed. To disable compression of the data, specify "enable" as 0.

Default is no compression.

***RETURN VALUES***

None

***SEE ALSO***

tiff_writefile

---

## *set_tiff_image_number*

***NAME***

set_tiff_image_number - specify the image to be read from a TIFF file

***SYNOPSIS***

```
#include "im_proto.h"

void set_tiff_image_number(int number)
```

***DESCRIPTION***

To read a sub-image that can be present in a TIFF file, set the number to the required image in the file. 1 is the first image in a TIFF file (and the default). If a number is specified that is not present in a TIFF file, the first image from that TIFF file will be taken.

***RETURN VALUES***

None

***SEE ALSO***

tiff_readfile

---

## *set_var_object_class*

***NAME***

set_var_object_class - change the class of an var_object

***SYNOPSIS***

```
#include "objectsp.h"

int set_var_object_class(VAR_OBJECT *obj, char *class)
```

***DESCRIPTION***

Changes the class of the object "obj" to "class". This allows you to maintain different groups of objects since is possible in the command description file of SCIL_Image to specify which class you want to show in the dialog-box. The selection is made by comparing the specified class with the field "class" in the var_object structure. Only the objects with a matching class name are shown.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

var_object  destroy_var_object  show_var_object_info  list_var_objects

---

## *set_var_object_comment*

***NAME***

set_var_object_comment - add a comment string to the var_object

***SYNOPSIS***

```
#include "objectsp.h"

int set_var_object_comment(VAR_OBJECT *obj, char *comment)
```

***DESCRIPTION***

Add a (null-terminated) string "comment" to the structure of the var_object "object". The string may be of any length as long as it is null-terminated. The function itself allocates memory for the string, so if adding comment to a var_object while not using this function, be sure that the memory was allocated with malloc() because other functions rely on it (they use free()). When the var_object is saved, the comment string will be saved in the header-file.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

var_object  destroy_var_object  show_var_object_info

---

## *set_var_object_data*

***NAME***

set_var_object_data - set the data of a var_object to type and sizes

***SYNOPSIS***

```
#include "objectsp.h"

int set_var_object_data(VAR_OBJECT *obj, int type, int nr_channels,
int nr_dim, int dim1, int dim2, int dim3, int dim4, int dim5)
```

***DESCRIPTION***

Set the data of a var_object to the desired type and sizes. This function does the same as set_var_object_type() and set_var_object_size() together. The var_object is set to type "type" and to the dimensions specified by "nr_channels", "nr_dim" and "dim1" .. "dim5". For a complete description on these parameters see the functions set_var_object_type() and set_var_object_size().

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

var_object  set_var_object_type  set_var_object_size

## *set_var_object_size*

***NAME***

set_var_object_size - change the sizes of a var_object

***SYNOPSIS***

```
#include "objectsp.h"

int set_var_object_size(VAR_OBJECT *obj, int nr_channels, int nr_dim,
int dim1, int dim2, int dim3, int dim4, int dim5)
```

***DESCRIPTION***

Changes the sizes of the object "obj". "nr_channels" is the number of channels for the object, "nr_dim" is the desired number of dimensions and "dim1" .. "dim5" specify the size of each dimension. The maximum number of dimensions is 5. If less than 5 dimensions are required then it is sufficient to specify only "nr_dim" dimensions in the function-call. The dimensions over "nr_dim" are set to 1, regardless whether they were supplied or not.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

var_object  set_var_object_data  set_var_object_type

## *set_var_object_type*

***NAME***

set_var_object_type - set the type of the data in a var_object

***SYNOPSIS***

```
#include "objectsp.h"

int set_var_object_type(VAR_OBJECT *obj, int type_of_data)
```

***DESCRIPTION***

Set the data-type of the var_object "obj" to the type "type_of_data". Only the following data-types are allowed:

| | |
|---|---|
| PIXEL_T | 1 |
| CHAR_T | 2 |
| SHORT_T | 4 |
| INT_T | 8 |
| LONG_T | 16 |
| FLOAT_T | 32 |
| DOUBLE_T | 64 |

The dimensions of the var_object remain the same, only the type changes. Because a new piece of memory is allocated for the changed type, the contents of the var_object will be lost(except of course if the data-type remains the same).

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

var_object  set_var_object_data  set_var_object_size

## *set_window_pos*

***NAME***

set_window_pos - set the position of the display window of an image

***SYNOPSIS***

```
#include "disp_p.h"

int set_window_pos(IMAGE *im, int x, int y)
```

***DESCRIPTION***

set_window_pos() puts the display window of the image "im" at position ("x","y").

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

set_start_pos  set_window_size  natural_window_size

---

## *set_window_size*

***NAME***

set_window_size - set the size of a display window

***SYNOPSIS***

```
#include "disp_p.h"

int set_window_size(IMAGE *im, int sizex, int sizey)
```

***DESCRIPTION***

set_window_size() changes the size of the display window of the image "im" to "sizex"*"sizey".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

natural_window_size  set_window_pos  set_start_pos

---

## *setbuf*

## *setbuffer*

## *setlinebuf*

## *setvbuf*

***NAME***

setbuf, setbuffer, setlinebuf, setvbuf - assign buffering to a stream

***SYNOPSIS***

```
#include <stdio.h>

void setbuf(FILE *stream, char *buf)

void setbuffer(FILE *stream, char *buf, int size)

void setlinebuf(FILE *stream)

int setvbuf(FILE *stream, char *buf, int type, int size)
```

***DESCRIPTION***

These functions are interface functions to the standard C functions as implemented on the current operating system. The functionality of these functions is:

The three types of buffering available are unbuffered, block buffered, and line buffered. When an output stream is unbuffered, information appears on the destination file or terminal as soon as written; when it is block buffered many characters are saved up and written as a block; when it is line buffered characters are saved up until a NEWLINE is encountered or input is read from stdin. fflush() (see fclose) may be used to force the block out early. Normally all files are block buffered. A buffer is obtained from malloc upon the first getc() or putc() on the file. If the standard stream stdout refers to a terminal it is line buffered. The standard stream stderr is unbuffered by default.

setbuf() can be used after a stream has been opened but before it is read or written. It causes the array pointed to by "buf" to be used instead of an automatically allocated buffer. If buf is the NULL pointer, input/output will be completely unbuffered. A manifest constant BUFSIZ, defined in the <stdio.h> header file, tells how big an array is needed:

```
      char buf[BUFSIZ];
```

setbuffer(), an alternate form of setbuf(), can be used after a stream has been opened but before it is read or written. It uses the character array buf whose size is determined by the size argument instead of an automatically allocated buffer. If buf is the NULL pointer, input/output will be completely unbuffered.

setvbuf() can be used after a stream has been opened but before it is read or written. type determines how stream will be buffered. Legal values for type (defined in <stdio.h>) are:

| | |
|---|---|
| _IOFBF | fully buffers the input/output. |
| _IOLBF | line buffers the output; the buffer will be flushed when a NEWLINE is written, the buffer is full, or input is requested. |
| _IONBF | completely unbuffers the input/output. |

If "buf" is not the NULL pointer, the array it points to will be used for buffering, instead of an automatically allocated buffer. "size" specifies the size of the buffer to be used.

setlinebuf() is used to change the buffering on a stream from block buffered or unbuffered to line buffered. Unlike setbuf(), setbuffer(), and setvbuf(), it can be used at any time that the file descriptor is active.

A file can be changed from unbuffered or line buffered to block buffered by using freopen() (see fopen). A file can be changed from block buffered or line buffered to unbuffered by using freopen() followed by setbuf() with a buffer argument of NULL.

***RETURN VALUES***

If an illegal value for type or size is provided, setvbuf() returns a non-zero value. Otherwise, the value returned will be zero.

***SEE ALSO***

fclose fopen fread getc malloc printf putc puts setbuf

## *SetMacFileType*

## *SetMacFileCreator*

***NAME***

SetMacFileType, SetMacFileCreator - set Macintosh file type/creator

***PLATFORM***

Macintosh

***SYNOPSIS***

```
#include "support.h"

void SetMacFileType(long type)

void SetMacFileCreator(long creator)
```

***DESCRIPTION***

On Macintosh systems, each file is assigned a so-called type and a creator. These four-characters strings are used by the operating system and application programs to identify which program created which data-file and classify the data-files into different categories. E.g. plain text files all have the type 'TEXT' so editors will only allow you to open files with type 'TEXT'.

By calling SetMacFileType() and SetMacFileCreator() you will set two global variables that hold these values to be used in all following calls to creat(), fopen() etc. until you assign a different value. However when not specifying binary mode when creating a file, some compilers will overrule the type with 'TEXT'.

***NOTE***

Please note the special notation for these strings, they are declared to be long int variables (4 bytes). The string should be specified using single quotes (') around the text e.g. 'TEXT'. Although single quotes may only used on single characters in C, Macintosh C-compilers will allow this notation.

***RETURN VALUES***

None

## *setprompt*

***NAME***

setprompt - Set scil command prompt

***PLATFORM***

UNIX.

***SYNOPSIS***

```
void setprompt(char *prompt)
```

***DESCRIPTION***

setprompt changes the scil default prompt "[C2]" to the specified prompt. In the prompt string an exclamation mark may be used to specify the position of the command number.

***EXAMPLE***

To set the scil prompt to "<hello 2>"

```
setprompt("<hello !>");
```

***RETURN VALUES***

None

## *sfp*

***NAME***

sfp - simulated fluorescence process (creates shadow images).

***SYNOPSIS***

```
#include "im_proto.h"

int sfp(IMAGE *In, IMAGE *Out, int Orientation, int Background, int
Light, int View, int Excitation, int Emission, int Extra_light)
```

***DESCRIPTION***

Creates a 2D image from a 3D image suggesting depth by means of shadows. The algorithm is based on Simulation of a Fluorescence Process. The object is lit (excited) by a imaginary source of light and as a result, shadows are projected on an artificial background. The direction from the exciting light as well as the direction from which you are viewing the object can be specified independently. A list of the parameters is given containing the meaning and the valid value intervals:

| parameter | interval | |
|---|---|---|
| Orientation | [0,1] | specifies whether the light and viewing direction is west-east (0) or north-south (1). |
| Background | [0,65535] | sets the value of the color of the background. |
| Light | [-3,3] | sets the direction the light is shining. The number is the angle of the light in voxels per slice. |
| View | [-3,3] | specifies the direction of view. |
| Excitation | [0,1.4] | specifies the absorption factor of the illuminating light by the voxels. A value of "1" means that a ray is attenuated by a factor 1/e if it passes through a voxel with value 255. |
| Emission | [0.04,1.4] | specifies the emission factor for the light that is sent back to the viewer. Works the same way as "Excitation". |
| Extra_light | [-4,3] | switches on an extra source of light from the specified direction. The intensity is half of that of the original source of light. It is used to light the parts of an object that are in the shadow of another part of the object. A value of -4 means no extra light source is used. |

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

## *shape*

### *NAME*

shape - calculate shape parameters of objects

### *SYNOPSIS*

```
#include "im_proto.h"

int shape(IMAGE *label_im, VAR_OBJECT *label, VAR_OBJECT *xcentre,
VAR_OBJECT *ycentre, VAR_OBJECT *area, VAR_OBJECT *peri, VAR_OBJECT
*p2a, VAR_OBJECT *ccoun, VAR_OBJECT *count, int nosmooth)
```

### *DESCRIPTION*

Calculate a number of shape parameters of objects. The image "label_im" is scanned from the upper left corner to the lower right corner. If a non-zero pixel is found within the image, this indicates the upper left corner of a labeled object. The measured parameters for each object will be stored into an output array (that is stored in a var_object). There is an output array for each parameter. The order of parameters in the arrays corresponds with the order in which the objects are detected. All measures except "label" and "p2a" are expressed in pixels. The following information will be stored in the output arrays:

| | |
|---|---|
| Label number | store the label number of the detected object into the element of the integer array "label" corresponding with the sequence number of the detected object. |
| Xcentre | store the X coordinate of the object's center of gravity into the floating point array "xcentre". |
| Ycentre | store the Y coordinate of the object's center of gravity into the floating point array "ycentre". |
| Area | store the area of the detected object into the integer array "area". |
| Perimeter | store the length of the curve, consisting of the (8-connected) contour of the object into the floating point array "peri". |
| P2a | store the value of the squared contour length divided by 4*PI times the area of the detected object, into the floating point array "p2a". In this measure, a curve following the outer contour of the object is used for calculation of the contour length. Note that this is not the same value as "peri", which is the length of a curve following the centers of the contour pixels. |
| Contour count | store the number of contour pixels of the detected object into the integer array "ccoun". |

The number of objects found will be stored into the scalar variable "count".

In general, before measuring the parameters, a local object contour smoothing is done on the obtained intermediate contour codes. The smoothing is performed by replacement of chain-code sequences which describe sharp corners or jagged edges by smoother chain-codes. This local contour smoothing is performed until no more changes occur. The smoothing may be suppressed by specifying the parameter "nosmooth" (=1).

The maximum number of values that can be stored in an array is 1024, so no more than 1024 object are measured in one image.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

density  calibrated_density  label

---

## *shift_im*

***NAME***

shift_im - pixel wise shift of image pixels

***SYNOPSIS***

```
#include "im_proto.h"

int shift_im(IMAGE *in, IMAGE *out, int nshift)
```

***DESCRIPTION***

Shift all pixels of image "in" "nshift" bits and store the result in image "out"

***NOTE***

For more powerful image arithmetic expressions (scaling, adding offsets, etc.), use the function eval() .

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

and_im  or_im  xor_im  invert_im

---

## *show_cur_dir*

***NAME***

show_cur_dir - Show directory in title bar.

***PLATFORM***

UNIX.

***SYNOPSIS***

```
void show_cur_dir(int flag)
```

***DESCRIPTION***

show_cur_dir enables/disables the display of the current working directory in the title bar of the command window. "flag" = 1 is enable, "flag" = 0 is disable

***RETURN VALUES***

None

---

## *show_dmode_flags*

***NAME***

show_dmode_flags - show the display mode flags of an image

***PLATFORM***

UNIX, Macintosh.

***SYNOPSIS***

```
#include "disp_p.h"

int show_dmode_flags(IMAGE *image)
```

***DESCRIPTION***

show_dmode_flags() displays on the terminal window, the value of the display mode flags of "image". This value includes the modes that can be set with both set_display_mode() and set_dither_mode(). The corresponding value can be found in the include file "dmodes". The different modes are all represented by a unique bit in a long word that is present in the viewport structure attached to the IMAGE structure.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

get_display_mode  set_display_mode  set_dither_mode

---

## *show_func_overload*

***NAME***

show_func_overload - show (part of) the function overload tables

***SYNOPSIS***

```
#include "im_infra.h"

int show_func_overload(char *spec_func, int im_type, char *file_name)
```

***DESCRIPTION***

show_func_overload can be used to display (part of) the function overload tables. The output looks like:

```
            im_type1    im_type2    im_type3
func_name   X           X
```

This means that function "func_name" is overloadable for image types "im_type1" and "im_type3" but not for type "im_type2".

If a "spec_func" is given only that function is taken into account, NULL as a first argument means all functions are taken into account.

If an "im_type" is given as the second parameter only functions that are overloadable for that image type are displayed (0 == all types).

If a "file_name" is given as the third argument the output is send to that file, with NULL the output is send to the controlling terminal.

***RETURN VALUES***

None

***SEE ALSO***

init_func_overload  overloadable_func

## *show_func_stack*

## *show_error_stack*

## *im_clear_func_stack*

## *im_clear_errors*

## *get_im_error_stack*

## *im_get_func_stack_copy*

***NAME***

show_func_stack, show_error_stack, im_clear_func_stack, im_clear_errors, get_im_error_stack, im_get_func_stack_copy - examination and manipulation of function and error-stack.

***SYNOPSIS***

```
#include "im_error.h "

void show_func_stack(void)

void show_error_stack(void)

void im_clear_func_stack(void)

void im_clear_errors(void)

void *get_im_error_stack(void)

void im_get_func_stack_copy(IM_FUNC_STACK *fstack, int *flevel)
```

***DESCRIPTION***

| | |
|---|---|
| fstack | - pointer to local storage for the func-stack |
| flevel | - pointer to local storage for the stack-level |

show_func_stack() performs a textual dump of the function stack that is maintained by the im_begin_func() and im_end_func() functions.

show_error_stack() performs a textual dump of the error-stack.

Both show_func_stack() and show_error_stack() use the image_output() function to display their information. So it is up to the user interface if and how this information is shown.

clear_func_stack() and clear_error_stack() erase the contents of the function- and error-stack. This function should only be used by an interface after an error has occurred and has been reported to the user.

get_im_error_stack() retrieves the pointer to the global error object "im_error_stack". This pointer can then be used to subscribe to the error-stack and thus receive notification if an error occurred in the image-processing.

im_get_func_stack_copy() copies the global function-stack and the stack-level counter to local storage.

Every user-interface for Image should take care of reporting errors to the user.

***RETURN VALUES***

None

***SEE ALSO***

im_begin_func im_end_func im_report_error image_output

---

## *show_image_info*

***NAME***

show_image_info - display information of an image

***SYNOPSIS***

```
#include "im_infra.h"

void show_image_info(IMAGE *im)
```

***DESCRIPTION***

show_image_info() displays the data of the IMAGE data-structure of the specified image "im" on the controlling terminal. This function is intended for debug purposes only.

***RETURN VALUES***

None

---

## *show_menu_layout*

***NAME***

show_menu_layout - dump the layout of the menu to file

***PLATFORM***

Unix.

***SYNOPSIS***

```
#include "md_gen.h"

int show_menu_layout(char *filename, int show_items)
```

***DESCRIPTION***

show_menu_layout() dumps the layout of the menu-system to a file by the name "filename". The "show_items" flag determines if the menu-item are to be dumped as well. "show_items" = 1 shows the menu and what is in the menus. "show_items" = 0. shows only the menus.

The output is formatted with spaces to reflect the menu hierarchy.

***RETURN VALUES***

None

## *show_statistics*

## *perc_to_pixel*

***NAME***

show_statistics - calculates statistic values of an image

perc_to_pixel - converts a percentage to a pixel value.

***SYNOPSIS***

```
#include "im_proto.h"

int show_statistics(IMAGE *in, int mode, int slicenr, int lo_limit,
int hi_limit)

int perc_to_pixel(IMAGE *in, int top, int mode, int slicenr, double
perc)
```

***DESCRIPTION***

show_statistics() calculates and prints the following values from the total image "in" or from a Z-slice "slicenr" of the 3D image "in" between the values ["lo_limit", "hi_limit"] :

| | |
|---|---|
| Histo size | size of internally calculated histogram, this is the same as the amount of pixel values. |
| Range | the interval that contains the voxel values of the image, can be influenced by lo_limit and hi_limit. |
| Range (perc.) | the offset percentage of the range, measured from (in percents) the top and the bottom of the histogram. |
| Amount | the number of voxels between lo_limit and hi_limit. |
| Mean value | the mean value of the histogram. 1/N sum_i (histo[i]*i) |
| Standard Dev | the standard deviation of the histogram. sqrt(1/N sum_i (histo[i]*i - mean)^2) |
| Specific Dev | the specific deviation. standard dev / modal |
| Std interval | the standard interval.[mean - standard dev, mean + standard dev] |
| RMS | the root mean square. 1/N sum_i (histo[i]*i*i) |
| Modal | the modal value. |
| left 1/e | the point left from modal with the frequency of 1/e*f_modal. This value can be out of range. |

| | |
|---|---|
| right 1/e | the point right from modal with the frequency of 1/e*f_modal. This value can be out of range. |
| Median | the 50% point in the histogram |
| 1st quartile | the 25% point in the histogram |
| 3rd quartile | the 75% point in the histogram |
| Background | the value between the background noise and the object data. This value is determined by searching the zero point in the 3rd derivation of the histogram. This will give the value of the lowest point between the background peak and the object peak. |
| Object perc | the percentage of object-voxels in the whole range. |

Percentile points can be calculated with the function perc_to_pixel(), it converts a percentage to a pixel value, by defining the histogram of the image "in" or slice (only 3D), and calculating the number of pixels from the top or the bottom of the histogram. "top" is "Top" (0) means calculating the percentage from the top of the histogram, "Bottom" (1) means from the bottom.

"mode" speci fies whether the entire image "in" is to be used ("Image" (0)) or just a slice ("Slice" (1)). "slicenr" tells which slice to use in "Slice" mode.

***RETURN VALUES***

perc_to_pixel() returns the pixel value at the percentage on success.
show_statistics() returns IE_OK (1) on success.
Negative error status on failure (see im_error.h)

## *show_var_object_info*

***NAME***

show_var_object_info - display information on a var_object

***SYNOPSIS***

```
#include "objectsp.h"

int show_var_object_info(VAR_OBJECT *obj)
```

***DESCRIPTION***

This function displays information on the var_object "obj" on the controlling terminal. The information displayed concerns the name, class, type and the dimensions of the object. Also the address of the data is displayed

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

list_var_objects  var_object  destroy_var_object  var_object_by_name

---

## *sigma*

***NAME***

sigma - standard deviation filter

***SYNOPSIS***

```
#include "im_proto.h"

int sigma(IMAGE *in, IMAGE *out, int fsize)
```

***DESCRIPTION***

Standard deviation filter with sizes "fsize" * "fsize" from image "in" to image "out".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

---

## *sign_im*

***NAME***

sign_im - sign

***SYNOPSIS***

```
#include "im_proto.h"

int sign_im(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

Determine the sign of each element of the image "in" and store the result in the corresponding element of the image "out". If an element of "in" has a positive or zero value, the result is 1, otherwise the result is -1.

***NOTE***

For more powerful image arithmetic expressions (scaling, adding offsets, etc.), use the function eval() .

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

negation_im

## ***silo_list***

***NAME***

silo_list - list contents of the silo entries

***SYNOPSIS***

```
#include "silo.h"

void silo_list(SILOPTR siloptr)
```

***DESCRIPTION***

| | | |
|---|---|---|
| siloptr | - | Pointer to the image-silo. |

Prints a list of all occupied entries. The entry information is printed:

entry number (silo_key).
start position in file.
sizex.
sizey.

***NOTE***

This routine is only used as a debugging tool. The routine will be removed in the future.

***RETURN VALUES***

None

## *silo_to_comp*

### *NAME*

silo_to_comp - transform image-silo into composite photo

### *SYNOPSIS*

```
#include "silo.h"

int silo_to_comp(SILOPTR siloptr, COMPTR comptr, int startlabel, int
endlabel)
```

### *DESCRIPTION*

| | | |
|---|---|---|
| siloptr | - | Pointer to an image-silo. |
| comptr | - | Pointer to an composite photo. |
| startlabel | - | Silo-entry to start with. |
| endlabel | - | Silo-entry to end with. |

Transfers a part of an image-silo to a composite photo. If the startlabel is bigger than the endlabel then the silo is scanned backwards.

### *NOTE*

It does not create a new composite photo but merely adds the silo to an already started composite photo.

### *RETURN VALUES*

IE_OK (1) on succes
Negative error status on failure (see im_error.h)

## *sin*

## *cos*

## *tan*

## *asin*

## *acos*

## *atan*

## *atan2*

## *sinh*

## *cosh*

## *tanh*

***NAME***

sin, cos, tan, asin, acos, atan, atan2, sinh, cosh, tanh - trigonometric functions

***SYNOPSIS***

```
#include <math.h>

double sin(double x)

double cos(double x)

double asin(double x)

double acos(double x)

double atan(double x)

double atan2(double x, double y)

double sinh(double x)

double cosh(double x)

double tanh(double x)
```

***DESCRIPTION***

These functions are interface functions to the standard C functions as implemented on the current operating system. The functionality of these functions is:

sin(), cos() and tan() return trigonometric functions of radian arguments. The magnitude of the argument should be checked by the caller to make sure the result is meaningful.

asin() returns the arc sin in the range -pi/2 to pi/2.

acos() returns the arc cosine in the range 0 to pi.

atan() returns the arc tangent of x in the range -pi/2 to pi/2.

atan2() returns the arc tangent of x/y in the range -pi to pi.

sinh(), cosh(), and tanh() compute the designated hyperbolic functions.

***RETURN VALUES***

Arguments of magnitude greater than 1 cause asin and acos to return value 0; errno is set to EDOM. The value of tan at its singular points is a huge number, and errno is set to ERANGE.

## *sin_im*

## *cos_im*

## *tan_im*

## *asin_im*

## *acos_im*

## *atan_im*

## *atan2_im*

## *sinh_im*

## *cosh_im*

## *tanh_im*

***NAME***

sin_im, cos_im, tan_im, asin_im, acos_im, atan_im, atan2_im, sinh_im, cosh_im, tanh_im - trigonometric functions on images

***SYNOPSIS***

```
#include "im_proto.h"

int sin_im(IMAGE *in, IMAGE *out)

int cos_im(IMAGE *in, IMAGE *out)

int tan_im(IMAGE *in, IMAGE *out)

int asin_im(IMAGE *in, IMAGE *out)

int acos_im(IMAGE *in, IMAGE *out)

int atan_im(IMAGE *in, IMAGE *out)

int sinh_im(IMAGE *in, IMAGE *out)

int cosh_im(IMAGE *in, IMAGE *out)

int tanh_im(IMAGE *in, IMAGE *out)

int atan2_im(IMAGE *in1, IMAGE *in2, IMAGE *out)
```

***DESCRIPTION***

Trigonometric functions to be performed on images. The extension "_im" is added because the trigonometric functions are predefined in C and therefore they cannot be used for whole images. If illegal operations are performed on pixels in the image, a count of the number of illegal operations is printed.

sin_im(), cos_im() and tan_im() return trigonometric functions of radian arguments.

asin_im() returns the arc sin in the range -pi/2 to pi/2.

acos_im() returns the arc cosine in the range 0 to pi.

atan_im() returns the arc tangent of x in the range -pi/2 to pi/2.

atan2_im() returns the arc tangent of in1/in2 in the range -pi to pi.

sinh_im(), cosh_im() and tanh_im() calculate the hyperbolic function of each element of the input image. The elements are assumed to be expressed in radians.

***RETURN VALUES***

asin_im, acos_im, atan2_im returns the number of illegal operations which occurred. Therefore these functions return 0 on success.

sin_im, cos_im, tan_im, atan_im, sinh_im, cosh_im and tanh_im return IE_OK (1) on success or negative error status on failure (see im_error.h)

---

## *single_pixels*

***NAME***

single_pixels - single point detection

***SYNOPSIS***

```
#include "im_proto.h"

int single_pixels(IMAGE *in, IMAGE *out, int bound, int conn, int
obj_bkg, int detect_rem)
```

***DESCRIPTION***

Detects single object pixels in image "in" and stores the result in image "out". The image is scanned by a moving window with dimensions 3*3. If the central pixel within the window is the only object pixel in the window, it keeps its value and is detected as a single object pixel. Otherwise the central pixel becomes a background pixel. "obj_bkg" specifies the kind of point that are searched for, object (1) or background (0). "detect_rem" specifies whether the points found should be detected (1) or removed (0). "bound" specifies that the edge around the image must be set to foreground (1) or to background (0) pixels. "conn" specifies the connectivity and can either be 4 or 8.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

---

## *size*

***NAME***

size

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See objectsize

---

## *skelpoints*

***NAME***

skelpoints - detect the special skeleton points

***SYNOPSIS***

```
#include "im_proto.h"

int skelpoints(IMAGE *in, IMAGE *out, int bound, int opcode, int
type)
```

***DESCRIPTION***

Detects special points in a skeleton in image "in" and stores the result in image "out". The type of points detected depends upon "opcode":

| | |
|---|---|
| 1 | end pixels |
| 2 | link pixels |
| 3 | branch pixels |

The image is scanned by a moving window with dimensions 3*3. In each window the central pixel is checked for being a special point. If so the central pixel remains an object pixel, otherwise it is deleted as an object pixel.

"type" determines how the skeleton points are defined; according to Hilditch (0) or according to Preston (1).

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

hild_skelet  holt_skelet

---

## *sleep*

***NAME***

sleep - suspend execution for interval

***SYNOPSIS***

```
unsigned sleep(unsigned seconds)
```

***DESCRIPTION***

This function is an interface to the standard C function as implemented on the current system. The functionality of this function is:

The current process is suspended from execution for the number of seconds specified by the argument. The actual suspension time may be up to 1 second less than that requested, because scheduled wakeups occur at fixed 1-second intervals, and an arbitrary amount longer because of other activity in the system.

The routine is implemented by setting an alarm clock signal and pausing until it occurs. The previous state of this signal is saved and restored. If the sleep time exceeds the time to the alarm signal, the process sleeps only until the signal would have occurred, and the signal is sent 1 second later.

---

## *small_object_removal*

***NAME***

small_object_removal - object select on size

***SYNOPSIS***

```
#include "im_proto.h"

int small_object_removal(IMAGE *in, IMAGE *out, int size)
```

***DESCRIPTION***

Calculate for each object in image "in" the object size by counting the number of pixels and compare the size with "size". If the size is less than "size", all object pixels of the object are deleted (replaced by 0). If the size is greater than or equal to size, the object is unmodified, i.e. it keeps its original label value.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

hull  label  objectsize  rhull

---

## *sobel_diff*

***NAME***

sobel_diff - Sobel edge operator

***SYNOPSIS***

```
#include "im_proto.h"

int sobel_diff(IMAGE *in, IMAGE *out, int mode)
```

***DESCRIPTION***

Differential edge detection based upon the Sobel operator. Within the moving window in image "in", with dimensions 3*3, the horizontal and vertical differential values are calculated by a convolution with the masks (horizontal respectively. vertical):

```
 1   2   1       -1   0   1
 0   0   0       -2   0   2
-1  -2  -1       -1   0   1
```

The output value is calculated from these convolutions, depending upon the "mode" specified:

| | |
|---|---|
| sqrt (1) | the output value is the square root of the sum of the quadratic convolution results |
| sum (0) | the output value is the sum of the absolute values of the convolution results |

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

prewitt_diff  roberts_diff  laplace  prewitt_temp  kirsch_temp  robinson_temp

---

## *sos*

***NAME***

sos

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See small_object_removal

---

## *spb_publish*

***NAME***

spb_publish - publish an event message to an object

***SYNOPSIS***

```
#include "spublish.h"

void spb_publish(void *obj, int mess, void *data)
```

***DESCRIPTION***

spb_publish() publishes the message "mess" to all subscribers of the object "obj". "data" is a pointer to object- and message-specific data that accompanies the event that triggered the publish. The type and contents of this data is determined by the combination of object-type and message. A description of this data must be retrieved from the documentation of the publishing object.

***NOTE***

The same message published by another type of object is very probably accompanied by other data, both in type and contents.

***RETURN VALUES***

None

***SEE ALSO***

spb_subscribe spb_unsubscribe

## *spb_subscribe*

## *spb_unsubscribe*

***NAME***

spb_subscribe - subscribe to an publishing object

spb_unsubscribe - unsubscribe from an publishing object

***SYNOPSIS***

```
#include "spublish.h"

void spb_subscribe(void *obj, void *id, SPBFUNC subscr, void *cldata)

void spb_unsubscribe(void *obj, void *id, SPBFUNC subscr)
```

***DESCRIPTION***

The function spb_subscribe() subscribes an object to another (publishing) object in order to receive notification of important events of that object by means of messages. "obj" is the publishing object. "id" is an identifier for the subscriber, typically a pointer to the subscribing object itself. "subscr" is a pointer to the function the subscriber uses to process the messages. "cldata" is an (optional) pointer to data the subscriber wants to receive whenever the publishing object broadcasts a message. The type and contents of this data is only known to the subscribing object, it is NOT interpreted by the subscribe-mechanism in any way.

spb_unsubscribe() removes the subscriber "id" from the list of subscriber of object "obj". "subscr" is a pointer to the function the subscriber used to process the messages. The function must be specified because a subscribing object can subscribe multiple functions to a single publishing object.

The message-processing function "subscr" must have the following function header:

```
void func(void *obj, void *id, int mess, void *data, void *cldata)
```

"obj" is the publishing object, "id" is the identifier for the subscribing object, "mess" is the message the publishing object broadcasts. "data" is pointer to the data that accompanies the message. "cldata" is a pointer to the data the subscribing object specified when subscribing to the publishing object.

***RETURN VALUES***

None

***SEE ALSO***

spb_publish

## *spix*

***NAME***

spix - pixel swapping

***SYNOPSIS***

```
#include "im_proto.h"

int spix(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

Swap the pixels of image "in" between even and odd columns and store the result into image "out". The effect of this operation is to mutually swap pairs of pixels that, in "packed" mode, were stored in one word. (Packed mode storage is usual for some peripherals and storage devices).

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

---

## *splih*

***NAME***

splih

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See split_horizontal

---

## *split*

***NAME***

split - image split-up

***SYNOPSIS***

```
#include "im_proto.h"

int split(IMAGE *in, IMAGE *out, int direct, int iter)
```

***DESCRIPTION***

Split image "in" into two halves, one half containing even lines and one half containing odd lines of image "in" and store the result in image "out". Depending on "direct", the command is executed on a per row (0) or per column (1) basis". The command is repeated "iter" times.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

split_horizontal  split_vertical  merge

## *split_channels*

## *join_channels*

***NAME***

split_channels - split a multi channel image in separate images

join_channels - build a multi channel image from separate images

***SYNOPSIS***

```
#include "im_proto.h"

int split_channels(IMAGE *in, IMAGE *out1, IMAGE *out2, IMAGE *out3,
IMAGE *out4)

int join_channels(IMAGE *in1, IMAGE *in2, IMAGE *in3, IMAGE *in4,
IMAGE *out, int im_type, int subtype)
```

***DESCRIPTION***

split_channels() takes each of the channels of a multi-channel image (up to 4 channels) and stores these channels in separate images. E.g. a RGB-color image is split in three images, one containing the red image, one the green image and one the blue image. Each of the output images "out1" .. "out4" may be a NULL pointer to retrieve only a subset of all the channels. The type of the output images is not changed by this function, they remain as they were before the call to split_channels().

join_channels() takes the images "in1" .. "in4" and constructs a multi-channel image from them. The type of the output image can be specified with the "im_type" parameter. If the image type has sub-types, e.g. RGB for color images, this can be set using "subtype". Each of the input images may be a NULL pointer to leave that channel untouched -if the output image is already a multi-channel image of the required "im_type" and "subtype"- or empty. In case a multi-channel image is chosen as an input image, the first channel of that image is taken; regardless of its position on the parameters list. The images "in1" .. "in4" must all be off the same size.

***NOTE***

split_channels() does not change the type of the output image in any way; they remain as they were before the call to the function.

***EXAMPLE***

uniform filtering a RGB-color image can be done by using a "normal" grey-value uniform filter on each of the channels.

```
#include "image.h"

readfile( "flamingo, a, 0,0);
split_channels(a, b, c, d, NULL );
uniform(b, b, 5, 5, 1);
uniform(c, c, 5, 5, 1);
uniform(d, d, 5, 5, 1);
join_channels(b, c, d, NULL, b, COLOR_2D, RGB_T);
```

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***
copy_channel

---

## *split_horizontal*

***NAME***
split_horizontal - split in horizontal direction

***SYNOPSIS***
```
#include "im_proto.h"

int split_horizontal(IMAGE *in, IMAGE *out, int iter)
```

***DESCRIPTION***
Same as split() with "direct" = 0.

***RETURN VALUES***
IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***
split_vertical split merge

---

## *split_vertical*

***NAME***
split_vertical - split in vertical direction

***SYNOPSIS***
```
#include "im_proto.h"

int split_vertical(IMAGE *in, IMAGE *out, int iter)
```

***DESCRIPTION***
Same as split() with "direct" = 1

***RETURN VALUES***
IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***
split_horizontal split merge

---

## *spliv*

***NAME***

spliv

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See split_vertical

---

## *sqrt_im*

***NAME***

sqrt_im - square root

***SYNOPSIS***

```
#include "im_proto.h"

int sqrt_im(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

Calculate the square root of each element of "in" and store the result in the corresponding element of "out".

***NOTE***

For more powerful image arithmetic expressions (scaling, adding offsets, etc.), use the function eval() .

***RETURN VALUES***

Number of domain conflicts (number of negative pixels in the input image), so 0 is OK.
Negative error status on failure (see im_error.h).

---

## *ssum*

***NAME***

ssum

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See pix_sum

---

## *start_comp*

***NAME***

start_comp - start a composite photo in an image

***SYNOPSIS***

```
#include "image.h"
#include "silo.h"

COMPTR start_comp(IMAGE *im)
```

***DESCRIPTION***

| | | |
|---|---|---|
| image | - | Image pointer. |

Function which must be called to start a composite photo in image "im". It returns a pointer to the composite photo. This pointer must be used in all subsequent composite photo references.

***RETURN VALUES***

Pointer to COMPOSIT struct
Negative error status on failure (see im_error.h)

## *stat*

## *fstat*

***NAME***

stat, fstat - get file status

***PLATFORM***

UNIX

***SYNOPSIS***

```
#include <sys/types.h>
#include <sys/stat.h>

int stat(char *name, struct stat *buf)

int fstat(int fildes, struct stat *buf)
```

***DESCRIPTION***

These functions are interface functions to the standard C functions as implemented on the current operating system. The functionality of these functions is:

stat() obtains detailed information about a named file. fstat() obtains the same information about an open file known by the file descriptor from a successful open(), creat(), dup() or pipe(2) call.

"name" points to a null-terminated string naming a file; "buf" is the address of a buffer into which information is placed concerning the file. It is unnecessary to have any permissions at all with respect to the file, but all directories leading to the file must be readable.

The layout of the structure pointed to by "buf" as defined in <stat.h> is given below. "st_mode" is encoded according to the "#define" statements.

```
struct stat {
      dev_t          st_dev;
      ino_t          st_ino;
      unsigned short st_mode;
      short          st_nlink;
      short          st_uid;
      short          st_gid;
      dev_t          st_rdev;
      off_t          st_size;
      time_t         st_atime;
      time_t         st_mtime;
      time_t         st_ctime;
};

#define S_IFMT   0170000 /* type of file */
#define S_IFDIR  0040000 /* directory */
#define S_IFCHR  0020000 /* character special */
#define S_IFBLK  0060000 /* block special */
#define S_IFREG  0100000 /* regular */
#define S_ISUID  0004000 /* set user id on execution */
#define S_ISGID  0002000 /* set group id on execution */
#define S_ISVTX  0001000 /* save swapped text even after use */
#define S_IREAD  0000400 /* read permission, owner */
```

```
#define S_IWRITE 0000200 /* write permission, owner */
#define S_IEXEC  0000100 /* execute/search permission, owner */
```

The mode bits 0000070 and 0000007 encode group and others permissions (see chmod(2)). The defined types, ino_t, off_t, time_t, name various width integer values; dev_t encodes major and minor device numbers; their exact definitions are in the include file <sys/types.h> (see types(5).

When fildes is associated with a pipe, fstat reports an ordinary file with restricted permissions. The size is the number of bytes queued in the pipe.

st_atime is the file was last read. For reasons of efficiency, it is not set when a directory is searched, although this would be more logical. st_mtime is the time the file was last written or created. It is not set by changes of owner, group, link count, or mode. st_ctime is set both by writing and changing the i-node.

***RETURN VALUES***

| | |
|---|---|
| 0 | is returned if a status is available; |
| -1 | if the file cannot be found. |

## *stereo_view*

***NAME***

stereo_view - calculate stereo images

***SYNOPSIS***

```
#include "im_proto.h"

int stereo_view(IMAGE *in, int mode, int view, IMAGE *left, IMAGE
*middle, IMAGE *right)
```

***DESCRIPTION***

stereo_view() calculates two or three images from which a stereo image can be made. It can calculate the front view and the views from 45 degrees to the left and 45 degrees to the right. "mode" specifies if the resulting images are kept in one output image or distributed over two or three image. When "mode" is "Join" (1) all the resulting images are put in the output image "left". "mode" is "Separate" (0) stores the results each in an image specified by "left", "middle" and "right". "view" determines which views are calculated, "LMR" (0) calculates all three views. "LM" (1) means the left and the middle front view, "MR" (2) the middle and right view and "LR" means the left and the right view.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

sfp

## *strcasecmp*

## *strncasecmp*

## *strsave*

***NAME***

strcasecmp, strncasecmp, strsave - string operations

***SYNOPSIS***

```
#include "support.h"

int strcasecmp(const char *str1, const char *str2)

int strncasecmp(const char *str1, const char *str2, size_t num)

char *strsave(const char *str)
```

***DESCRIPTION***

strcasecmp() and strncasecmp() compare two strings in a case-insensitive way. Their behavior is like the ANSI-C strcmp() and strncmp() functions except that differences in upper and lower case in the strings are ignored. If the strings are equal, strcasecmp() returns zero. strncasecmp() returns zero if the strings are equal up to "num" characters.

strcasecmp() and strncasecmp() are two defines in the "support.h" that point to the functions str_casecmp() and strn_casecmp() respectively. This is done to prevent name-conflicts on platforms that supply strcasecmp() and strncasecmp() in the C-library.

strsave() copies the string "str" to newly allocated memory and returns a pointer to the new string. If the string is empty or no string is  supplied, NULL is returned.

***RETURN VALUES***

strcasecmp() and strncasecmp() return zero if the strings are equal, a negative value if "str1" is less than "str2" and a positive value if "str1" is greater than "str2".

strsave() returns a pointer the newly allocated string or NULL if it cannot allocate enough memory.

## *strcat*

## *strncat*

## *strcmp*

## *strncmp*

## *strcpy*

## *strncpy*

## *strlen*

## *strchr*

## *strrchr*

## *index*

## *rindex*

## *strspn*

## *strcspn*

## *strpbrk*

## *strstr*

## *strtok*

## *strerror*

### *NAME*

strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen, strchr, strrchr, index, rindex, strspn, strcspn, strpbrk, strstr, strtok, strerror - string operations

### *SYNOPSIS*

```
char *strcat(char *s1, char *s2)

char *strncat(char *s1, char *s2, int n)

int strcmp(char *s1, char *s2)

int strncmp(char *s1, char *s2, int n)

char *strcpy(char *s1, char *s2)

char *strncpy(char *s1, char *s2, int n)

unsigned int strlen(char *s)

char *strchr(char *s, char c)
char *strrchr(char *s, char c)

char *index(char *s, char c)

char *rindex(char *s, char c)

size_t strspn(char *s, char *t)

size_t strcspn(char *s, char *t)

char *strpbrk(char *s, char *t)

char *strstr(char *s, char *t)

char *strtok(char *s, char *t)

char *strerror(int errnum)
```

***DESCRIPTION***

These functions are interface functions to the standard C functions as implemented on the current operating system. The functionality of these functions is:

These functions operate on null-terminated strings. They do not check for overflow of any receiving string.

strcat() appends a copy of string "s2" to the end of string "s1". strncat() copies at most "n" characters. Both return a pointer to the null-terminated result.

strcmp() compares its arguments and returns an integer greater than, equal to, or less than 0, according as "s1" is lexicographically greater than, equal to, or less than "s2". strcmp() uses native character comparison, which is signed. strncmp() makes the same comparison but looks at most "n" characters.

strcpy() copies string "s2" to "s1", stopping after the null character has been moved. strncpy() copies exactly "n" characters, truncating or null-padding "s2"; the target may not be null-terminated if the length of "s2" is "n" or more. Both return "s1".

strlen() returns the number of non-null characters in "s".

strchr() (strrchr()) returns a pointer to the first (last) occurrence of character "c" in string "s", or zero if "c" does not occur in the string.

strspn() returns the length of the prefix of "s" that consists of characters from "t".

strcspn() returns the length of the prefix of "s" that consists of characters not in "t".

strpbrk() returns a pointer to the first occurrence in string "s" of any characters from string "t", or NULL if none are present.

strstr() returns a pointer to the first occurrence of string "t" in string "s", or NULL if not present strtok() searches string "s" for tokens delimited by characters from "t", A sequence of calls of strtok(s,t) splits "s" into tokens, each delimited by a characters from "t". The first call in a sequence has a non-NULL "s". It finds the first token in"s" consisting of characters not in "t"; it terminates that by overwriting the next character of "s" with ’\0’ and returns a pointer to the token. Each subsequent call, indicated by a NULL value of "s", returns the next such token, searching from just past the end of the previous one. strtok() returns NULL when no further token is found, the string "t" may be different on each call.

***SEE ALSO***

memcmp  memcpy  memchr

---

## *swab*

***NAME***

swab - swap bytes

***SYNOPSIS***

```
void swab(char *from, char *to, int nbytes)
```

***DESCRIPTION***

This function is an interface to the standard C function as implemented on the current system. The functionality of this function is:

swab() copies "nbytes" bytes pointed to by "from" to the position pointed to by "to", exchanging adjacent even and odd bytes. It is useful for carrying binary data between to other machines. "nbytes" should be even.

---

## *sync_display*

***NAME***

sync_display - wait for all window-system events to be processed

***PLATFORM***

UNIX

***SYNOPSIS***

```
#include "disp_p.h"

int sync_display(void)
```

***DESCRIPTION***

sync_display() causes the X window systems request buffer to be flushed and then waits for all window-system events to be processed by the X server.

This function is generally only needed for debugging interactive functions/applications during which you want to be sure that all window-system calls have been processed.

***RETURN VALUES***

IE_OK (1)

---

## *system*

***NAME***

system - issue a shell command

***PLATFORM***

UNIX, MS-Windows.

***SYNOPSIS***

```
int system(char *string)
```

***DESCRIPTION***

This function is an interface to the standard C function as implemented on the current system. The functionality of this function is:

system() causes the string to be given to shell as input as if the string had been typed as a command at a terminal. The current process waits until the shell has completed, then returns the exit status of the shell.

***RETURN VALUES***

Exit status 256 indicates the shell couldn’t be executed.

---

## *t_morphology*

***NAME***

t_morphology - threshold morphology transform

***SYNOPSIS***

```
#include "im_proto.h"

int t_morphology(IMAGE *in, IMAGE *out, IMAGE *se, int thr, int
bound)
```

***DESCRIPTION***

Threshold transform on the binary image "in" using the weighted structuring element "se" with the given threshold value "thr" and store the result in the binary image in "out".

The threshold transform is equivalent with a correlation of the binary image "in" (valued 1 for object pixels and 0 for background pixels) with the mask given in the grey value image "se". The result of this correlation (a grey value image) is then thresholded. If the correlation sum is greater than or equal to "thr" the corresponding pixel in the output image is an object pixel, else a background pixel. "bound" specifies if the pixels outside the image should be interpreted as foreground pixels ("bound" = 1) or as background pixels ("bound" = 0).

Threshold morphological transform can be used to detect specific structures in binary images. All the pixels in the structuring element with positive weight (grey value) ideally should be object pixels. All pixels with negative weight ideally should be background pixels. Pixels with weight zero are don’t care pixels.

As an example consider the following weighted structuring element to detect horizontal lines in an image:

| | | | | | | |
|---|---|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| +1 | +1 | +1 | +1 | +1 | +1 | +1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 |

Correlating this mask with a binary image the resulting value in a pixel is between -42 (the sum of all negative weighted pixels) and +7 (the sum of all positive weighted pixels). Note that in case the correlation sum equals +7 the structure defined with the structuring element exactly fits in the image. Thus setting the threshold value equal to +7 (or in general equal to the sum of all positive values) gives the hit-or-miss transform. In order to find the structure also when some noise is present we may lower the threshold to 5.

Because in the above defined structure there are far more background pixels than there are object pixels, a better choice for the weighted structuring element is:

| | | | | | | |
|---|---|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| +6 | +6 | +6 | +6 | +6 | +6 | +6 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 |

Note that in this case the correlation value ranges from -42 to +42. In a sense, with our choice of weights we indicate that an object pixel is 6 times as important as a background pixel. But also that all background pixels are equally important (as are the object pixels).

A further refinement is to take into account that pixels further from the center line are less important than those close to the center line. Thus a new (and better) choice for the weighted structuring element is:

| | | | | | | |
|---|---|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -2 | -2 | -2 | -2 | -2 | -2 | -2 |
| -4 | -4 | -4 | -4 | -4 | -4 | -4 |
| +14 | +14 | +14 | +14 | +14 | +14 | +14 |
| -4 | -4 | -4 | -4 | -4 | -4 | -4 |
| -2 | -2 | -2 | -2 | -2 | -2 | -2 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 |

In this case the correlation value ranges from -98 to +98.

Further refinements are possible. We can take into account that the line thickness may vary by introducing extra lines in our weighted structuring element with positive weights. Of course the structures to be detected need not be restricted to horizontal lines. Any geometrical structure can be used.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

arbit_erosion  arbit_dilation  hit_or_miss  real_time_recognizer

## *taylor_polynomial*

## *taylor_expansion*

***NAME***

taylor_polynomial, taylor_expansion - create and apply taylor polynomial

***SYNOPSIS***

```
#include "im_proto.h"

int taylor_polynomial(VAR_OBJECT *out, int order, double delta_x,
double delta_y, double delta_s)

int taylor_expansion(IMAGE *in, IMAGE *out, double sigma, int order,
double accuracy, double delta_x, double delta_y, double t)
```

***DESCRIPTION***

taylor_polynomial() creates the taylor expansion polynomial up to "order" derivatives" for spatial offset ("delta_x", "delta_y") and scale offset "delta_s". The coefficients in the output var_object "out" (of type DOUBLE_T) represents the factor which each derivative should be multiplied before summation. The organization of the output is as follows:

{L}, {Ly,Lx}, {Lyy, Lyx, Lxx}, …

Each factor belongs to the corresponding derivative in

{f}, {df/dy, df/dx}, {df/dydy, df/dydx, df/dxdx}, ...

The function taylor_expansion() applies the expansion to image "in", by means of gaussian derivative filters using "order" "sigma" and "accuracy". The scale parameter is replaced by -t*sqrt("sigma"), resulting in the natural scale for t=0.5.

***LITERATURE***

L. Florack, The syntactical structure of scalar images, PhD Thesis, University of Utrecht, The Netherlands, 1991.

***RETURN VALUES***

IE_OK (1) on succes

Negative error status on failure (see im_error.h)

***SEE ALSO***

gauss_family  fuzzy_derivative

## *taylor_segmentation*

***NAME***

taylor_segmentation - segments a textured image by NJet examination

***SYNOPSIS***

```
#include "im_proto.h"

int taylor_segmentation(IMAGE *in, IMAGE *out, double dscale, int
order, double smooth)
```

***DESCRIPTION***

Performs a segmentation on the textured image "in" by means of differentiation. This can be interpreted as extracting the Taylor series or local NJet of the input image. The order of the taylor expansion is given by "order"; the scale for differentiation is "dscale". After the convolution, local energy is taken with "smooth" determining the gaussian sigma. A non-linear transform (sqrt_im) is applied and a feature reduction by means of Karhunen-Loeve Transform. The final result is stored in image "out".

***LITERATURE***

M. Unser and M. Eden, Nonlinear operators for improving texture segmentation based on features extracted by spatial filtering, IEEE Transactions on Systems, Man, and Cybernetics, vol. 20, 1990, 804-815.

J.J. Koenderink and A.J. van Doorn, Receptive field families, Biological Cybernetics, vol. 63, 1990, 291-297.

***RETURN VALUES***

IE_OK (1) on succes
Negative error status on failure (see im_error.h)

***SEE ALSO***

gauss_family  mul_im  vgauss  sqrt_im  karhunen_loeve

## *tcl_readfile*

***NAME***

tcl_readfile - read an image from a file in TCL format

***SYNOPSIS***

```
#include "im_proto.h"

IMAGE *tcl_readfile(char *filename, IMAGE *image, int xpos, int ypos)
```

***DESCRIPTION***

Read the image stored in file "filename" and put it in image "image". If "USE_NAME" (a NULL pointer) is specified as the image, a new image is created at position "xpos", "ypos", with the same name as the file. If an image is already present with that name, that image will be used.

The file must have the ".dat" extension.

***RETURN VALUES***

The pointer to the image in which the data was put, either an existing image or a newly created one.
NULL pointer on failure

***SEE ALSO***

readfile ics_readfile tiff_readfile aim_readfile jpeg_readfile writefile tcl_writefile

## *tcl_writefile*

***NAME***

tcl_writefile - write an image to file in TCL format

***SYNOPSIS***

```
#include "im_proto.h"

int tcl_writefile(IMAGE *image, char *filename)
```

***DESCRIPTION***

Write the image "image" to the file "filename" using the TCL format. If the image contains only 8 bit data (grey values 0..255), the image will be written in packed format (8 bit per pixel), otherwise the data will be written using 16 bits per pixel.

The file will have the ".dat" extension.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

writefile  ics_writefile  tiff_writefile  jpeg_writefile  readfile  tcl_readfile

## *threshold*

## *bi_threshold*

***NAME***

threshold - thresholding into binary image

bi_threshold - thresholding with two levels into binary image

***SYNOPSIS***

```
#include "im_proto.h"

int threshold(IMAGE *in, IMAGE *out, int level)

int bi_threshold(IMAGE *in, IMAGE *out, int low, int high)
```

***DESCRIPTION***

threshold performs a thresholding operation on the grey value image "in" and stores the result in the binary image "out". If the value of a pixel is greater than or equal to "level" the corresponding bit in the "out" image is set to "1". Otherwise it is set to "0".

bi_threshold() converts all the pixel in the range from "low" to "high" into "1" pixels in the output and all pixels that are outside that range to "0" pixels. "low" and "high" are converted to "1" pixels.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

clip  contrast_stretch  equalize  tri_state_threshold  lookup

## *ti_block*

***NAME***

ti_block - generate chess-board test image

***SYNOPSIS***

```
#include "im_proto.h"

int ti_block(IMAGE *out, int block_size, int forgr, int backgr)
```

***DESCRIPTION***

Generate a chess-board like pattern and store the result in "out". The pattern consists of alternating foreground and background squares. The size of the squares are given by "block_size". The pixel values of the fore- and background squares are given by "forgr" and "backgr".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

ti_circle  ti_hlines  ti_lines  ti_lshading  ti_points  ti_qshading  ti_vlines

## *ti_circle*

***NAME***

ti_circle - generate test image with concentric circles

***SYNOPSIS***

```
#include "im_proto.h"

int ti_circle(IMAGE *out, int dist, int rad, int line_w, int forg,
int back)
```

***DESCRIPTION***

Generate a pattern of concentric circles and store the result in "out". The center point of the circles will be at the center of the image. The distance between the circles is specified by "dist". The starting point of the innermost circle is specified "rad", measured from the center point. "line_w" specifies the width of the circles. "forg" and "back" specify the pixel values of the fore- and background.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

ti_block  ti_hlines  ti_lines  ti_lshading  ti_points  ti_qshading  ti_vlines

## *ti_fractal*

## *ti_ifr*

***NAME***

ti_fractal, ti_ifr - generate fractal images

***SYNOPSIS***

```
#include "im_proto.h"

int ti_fractal(IMAGE *out, double dim)

int ti_ifr(IMAGE *out, double slope)
```

***DESCRIPTION***

These functions generate fractals. ti_fractal() generates a fractal image with fractal dimension given by "dim". The function ti_ifr() generates an inverse function response, with slope "slope". After multiplying with a random image and applying a hartley or fourier transform, the spatial fractal is obtained. This function can be interpreted as a "fractal" point spread function.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

## *ti_hlines*

### *NAME*

ti_hlines - generate test image with horizontal lines

### *SYNOPSIS*

```
#include "im_proto.h"

int ti_hlines(IMAGE *out, int dist, int start_p, int line_w, int
forg, int back)
```

### *DESCRIPTION*

Generate a horizontal line pattern and store the result in image "out". The distance between the lines is specified by "dist". The starting point of the line pattern is specified by "start_p", measured from the top side of the image. "line_w" specifies the width of the lines. "forg" and "back" specify the pixel values of the fore- and background.

### *RETURN VALUES*

IE_OK (1) on success
Negative error status on failure (see im_error.h)

### *SEE ALSO*

ti_block ti_circle ti_lines ti_lshading ti_points ti_qshading ti_vlines

## *ti_lines*

***NAME***

ti_lines - generate test image with crossing lines

***SYNOPSIS***

```
#include "im_proto.h"

int ti_lines(IMAGE *out, int dist, int start_p, int line_w, int forg,
int back)
```

***DESCRIPTION***

Generate a pattern of crossing lines and store the result in image "out" The distance between the lines is specified by "dist". The starting point of the line pattern is specified by "start_p", measured from the top side and from the left side of the image. "line_w" specifies the width of the lines. "forg" and "back" specify the pixel values of the fore- and background.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

ti_block  ti_circle  ti_hlines  ti_lshading  ti_points  ti_qshading  ti_vlines

---

## *ti_lshading*

***NAME***

ti_lshading - generate linear shading image

***SYNOPSIS***

```
#include "im_proto.h"

int ti_lshading(IMAGE *out, double top_r, double top_l, double bot_l)
```

***DESCRIPTION***

Generate a linearly shaded grey value image and store the result in image "out". The intensity is defined by "top_r", "top_l" and "bot_l". "top_r" specifies the value of the top right pixel, "top_l" the top left pixel and "bot_l" the bottom left pixel.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

ti_block  ti_circle  ti_hlines  ti_lines  ti_points  ti_qshading  ti_vlines

---

## *ti_points*

***NAME***

ti_points - generate test image with symmetric points

***SYNOPSIS***

```
#include "im_proto.h"

int ti_points(IMAGE *out, int dist, int start_p, int width, int forg,
int back)
```

***DESCRIPTION***

Generate a pattern of equidistant square dots and store the result in image "out". The distance between the dots in horizontal and vertical direction is specified by "dist". The starting point of the pattern is specified by "start_p", measured from the top side and from the left side of the image. "width" specifies the dimensions of the dots. "forg" and "back" specify the pixel values of the fore- and background.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

ti_block  ti_circle  ti_hlines  ti_lines  ti_lshading  ti_qshading  ti_vlines

---

## *ti_qshading*

***NAME***

ti_qshading - generate quadratic shading image

***SYNOPSIS***

```
#include "im_proto.h"

int ti_qshading(IMAGE *out, int cval, int top_l)
```

***DESCRIPTION***

Generate a quadratically shaded grey value image and store the result in image "out". The intensity is defined by "cval" and "top_l". "cval" specifies the intensity at the center point and "top_l" the intensity in the top left pixel.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

ti_block  ti_circle  ti_hlines  ti_lines  ti_lshading  ti_points  ti_vlines

---

## *ti_vlines*

***NAME***

ti_vlines - generate test image with vertical lines

***SYNOPSIS***

```
#include "im_proto.h"

int ti_vlines(IMAGE *out, int dist, int start_p, int line_w, int
forg, int back)
```

***DESCRIPTION***

Generate a vertical line pattern and store the result in image "out". The distance between the lines is specified by "dist". The starting point of the line pattern is specified by "start_p", measured from the left side of the image. "line_w" specifies the width of the lines. "fore" and "back" specify the pixel values of the foreground and background.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

ti_block  ti_circle  ti_hlines  ti_lines  ti_lshading  ti_points  ti_qshading

---

## *ticb*

***NAME***

ticb

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See ti_block

---

## *ticc*

***NAME***

ticc

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See ti_circle

---

## *tiff_readfile*

***NAME***

tiff_readfile - read an image from a file in TIFF format

***SYNOPSIS***

```
#include "im_proto.h"

IMAGE *tiff_readfile(char *filename, IMAGE *image, int xpos, int
ypos)
```

***DESCRIPTION***

Read the image stored in file "filename" and put it in image "image". If "USE_NAME" (a NULL pointer) is specified as the image, a new image is created at position "xpos", "ypos", with the same name as the file. If an image is already present with that name, that image will be used.

The read function is capable of reading TIFF-files according to the TIFF 6.0 specifications. The file must have an extension that starts with ".tif". The extensions used for finding a TIFF file are ".tif" and ".tiff".

If more than one image is present in a TIFF file, the number of the image to be read can be set by the function "set_tiff_image_number".

***RETURN VALUES***

The pointer to the image in which the data was put, either an existing image or a newly created one.
NULL pointer on failure

***SEE ALSO***

set_tiff_image_number readfile ics_readfile tcl_readfile aim_readfile jpeg_readfile writefile tiff_writefile

## *tiff_writefile*

***NAME***

tiff_writefile - write an image to a file in TIFF format

***SYNOPSIS***

```
#include "im_proto.h"

int tiff_writefile(IMAGE *image, char *filename)
```

***DESCRIPTION***

Write the image "image" to the file "filename" using the TIFF format. The write function writes TIFF-files according to the TIFF 6.0 specifications.

The file will have the extension ".tif".

By default the data will be written uncompressed. To write the data compressed, use the function "set_tiff_compression".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

set_tiff_compression writefile ics_writefile tcl_writefile jpeg_writefile readfile tiff_readfile

---

## *tilh*

***NAME***

tilh

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See ti_hlines

---

## *tiln*

***NAME***

tiln

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See ti_lines

---

## *tils*

***NAME***

tils

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See ti_lshading

---

## *tilv*

***NAME***

tilv

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See ti_vlines

---

## *time*

***NAME***

time - time a command

***SYNOPSIS***

```
time <command>
```

***DESCRIPTION***

The command time records system and user time of a specific command. Typically useful in benchmarking. Anything can be timed.

***EXAMPLE***

```
[C1] int i=1000;
[C2] time while(i--);
user time: 0.983 system time: 0.000
[C3]
```

---

## *tipt*

***NAME***

tipt

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See ti_points

---

## *tiqs*

***NAME***

tiqs

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See ti_qshading

---

## *tmpfile*

## *tmpnam*

***NAME***

tmpfile, tmpnam - create temporary files

***SYNOPSIS***

```
#include <stdio.h>

FILE *tmpfile(void)

char *tmpnam(char *s)
```

***DESCRIPTION***

This function is an interface to the standard C function as implemented on the current system. The functionality of this function is:

tmpfile() creates a temporary file of mode "wb+" that will be automatically removed when closed or when the program terminates normally. tmpfile() returns a stream, or NULL, if it could not create the file.

tmpnam(NULL) creates a string that is not the name of an existing file, and returns a pointer to an internal static array. tmpnam(s) stores the string in "s" as well as returning it as the function value. "s" must" have room for at least "L_tmpnam" characters. tmpnam() generates a different name each time it is called ; at most TMP_MAX different names are guaranteed during execution of the program. Note that tmpnam() creates a name, not a file.

***RETURN VALUES***

See the description of the functions.

## *tolower*

## *toupper*

***NAME***

tolower, toupper - change the case of characters

***SYNOPSIS***

```
#include <ctype.h>

int tolower(int c)

int toupper(int c)
```

***DESCRIPTION***

This function is an interface to the standard C function as implemented on the current system. The functionality of this function is:

If "c" is an uppercase letter, tolower(c) returns the corresponding lowercase letter; otherwise it returns "c". If "c" is an lowercase letter, toupper(c) returns the corresponding uppercase letter; otherwise it returns "c".

***SEE ALSO***

islower  isupper

## *tri_state_threshold*

***NAME***

tri_state_threshold - 3-state thresholding

***SYNOPSIS***

```
#include "im_proto.h"

int tri_state_threshold(IMAGE *in, IMAGE *out, int thresh, int val1,
int fl1, int val2, int fl2, int val3, int fl3)
```

***DESCRIPTION***

Depending upon the threshold value "thres", change all pixel values of image "in" into one of the values "val1", "val2" or "val3", according to the rules:

- if input pixel < "thres": output pixel = "val1"
- if input pixel = "thres": output pixel = "val2"
- if input pixel > "thres": output pixel = "val3"

and store the result in image "out". If "fl1" is 0 and the first rule is applied, then the value of the output pixel will not be set to "val1" but to the value of the input pixel. The same applies to "fl2" and "fl3".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

clip threshold contrast_stretch equalize lookup mix

## *truncate_im*

***NAME***

truncate_im - truncate pixel values

***SYNOPSIS***

```
#include "im_proto.h"

int truncate_im(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

Convert each element of image "in" into an integer value by means of truncation of the fractional part and store the result into the corresponding elements of "out". For positive values the effect is that the integer value just less than or equal to the original value is taken. For negative values the effect is that the integer value just greater than or equal to the original value is taken.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

fraction_im  nearest_int  lowest_int

## *TWAcquire*

## *TWAcquireArea*

***NAME***

TWAcquire - acquire an image from a Twain device

TWAcquireArea - acquire a region of  an image from a Twain device

***PLATFORM***

MS-Windows.

***SYNOPSIS***

```
#include "image.h"

int TWAcquire(IMAGE *image)

int TWAcquireArea(IMAGE *image, double res, double xleft, double
ytop, double xright, ybottom, int type)
```

***DESCRIPTION***

TWAcquire() (=TwainAcquire in the menu) retrieves an image from a Twain compliant device that is attached to the system. The image is stored in "image". If more that one device is present, the default device is taken. Currently no provisions are available to select another than the default device.

The device must be attached to and its (driver) software properly installed on the system as controlling the device is done by the device software itself.

TWAcquireArea() (=TwainScan in the menu) retrieves a region of the image that can be acquired from an attached device. The size and position of the region to scan is determined by "xleft", "ytop", "xright" and "ybottom". The scan resolution is given by "res". The type of the data is specified by "type" and can be one of the following values:

| | |
|---|---|
| 0 | Unknown |
| 1 | Binary |
| 2 | Grey value |
| 3 | Color |

***RETURN VALUES***

1 on success
0 on failure

## *txt*

***NAME***

txt

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See image_text

---

## *unequal0_ok*

***NAME***

unequal0_ok - check to see if an integer value is unequal to zero

***SYNOPSIS***

```
#include "im_infra.h"

int unequal0_ok(int value, char *text)
```

***DESCRIPTION***

If the value "value" is not equal to zero, an error is generated and the following message is added to the error-stack:

```
<text> [<value>] must be unequal to 0
```

***NOTE***

This functions can only check on integer values, to check on floating point values, use the function funequal0_ok()

***RETURN VALUES***

IE_OK (1) if "value" is unequal to zero
IE_NOT_OK (0) if it is equal to zero

***SEE ALSO***

positive_ok  greater0_ok  funequal0_ok

---

## *ungetc*

***NAME***

ungetc - push character back into input stream

***PLATFORM***

UNIX.

***SYNOPSIS***

```
#include <stdio.h>

int ungetc(char *c, FILE *stream)
```

***DESCRIPTION***

This function is an interface to the standard C function as implemented on the current system. The functionality of this function is:

ungetc() pushes the character "c" back on an input stream. That character will be returned by the next getc() call on that stream. ungetc() returns c.

One character of pushback is guaranteed provided something has been read from the stream and the stream is actually buffered. Attempts to push EOF are rejected.

fseek() erases all memory of pushed back characters.

***RETURN VALUES***

ungetc returns EOF if it can’t push a character back.

***SEE ALSO***

getc  setbuf  fseek

## *uniform*

***NAME***

uniform - uniform linear filtering

***SYNOPSIS***

```
#include "im_proto.h"

int uniform(IMAGE *in, IMAGE *out, int filtx, int filty, int filtz)
```

***DESCRIPTION***

Image "in" is scanned with a moving window with sizes "filtx" * "filty" ("filtx" * "filty" * filtz" for 3D images). For each window position the average value of the pixels within the window is calculated. This average pixel value is stored into the pixel in image "out" that corresponds with the central pixel in the window.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

---

## *uniform_round*

***NAME***

uniform_round - uniform filter using a circular filter window

***SYNOPSIS***

```
#include "im_proto.h"

int uniform_round(IMAGE *in, IMAGE *out, int fsize, int norm)
```

***DESCRIPTION***

Image "in" is uniform filtered with a circular window and the result is put in image "out". The diameter of the circular window is "fsize". If "norm" is On (1) the result of the filter is normalized.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

uniform  grey_morph_round

---

## *update*

***NAME***

update - delete marked objects from an object list

***SYNOPSIS***

```
#include "im_aio.h"

LIST *update(LIST *list)
```

***DESCRIPTION***

| | | |
|---|---|---|
| list | - | List with objects |

update() removes all marked objects from the object list "list". Objects are marked from removal with the rm_object() function.

***NOTE***

The object is not removed from the image. You need to call hide_object() to remove an object from an image.

***EXAMPLE***

```
To remove objects touching the edge of an image:

#include "image.h"
#include "im_aio.h"
LIST *l, *o;

readfile("cermet",a,0,0);
threshold(a,b,128);
invert_im(b,b);
l = list_label(b,c,8,0);
FORALL(o,l) if(edge_object(c,o)) rm_object(o);
l = update(l);
/*
* Now to prove that the objects are no longer in the list
*/
FORALL(o,l) copy_object(c,d,o);
l = rm_list(l);
```

***RETURN VALUES***

pointer to the updated list

***SEE ALSO***

hide_object  rm_object

## *val_check*

***NAME***

val_check - check if a value is smaller than the image size

***SYNOPSIS***

```
#include "im_infra.h"
#include "command.h"

int val_check(IMAGE *image, int axis, int value)
```

***DESCRIPTION***

The function checks if "value" is positive and smaller than the size of the dimension specified by "axis". The values that "axis" can have are defined in the include file "command.h" and they are:

| | |
|---|---|
| WIDTH (1) | to check if "value" is smaller than the width of the image |
| HEIGHT (2) | to check if "value" is smaller than the height of the image |
| DEPTH (3) | to check if "value" is smaller than the width of the image |

A special value for "axis" if END (-1) that just returns the value -1. This is a special case implemented for the function im_val_ok() which performs a series of calls to val_check().

If the value is outside the image, for each of the different axis another message is generated, telling which axis is involved.

***RETURN VALUES***

END (-1) if "axis" is -1
IE_OK (1) if the value is within the image borders
IE_NOT_OK (0) if the value outside the image

***SEE ALSO***

im_val_ok  range_ok

## *var_object*

***NAME***

var_object - create a var_object

***SYNOPSIS***

```
#include "objectsp.h"

VAR_OBJECT *var_object(char *name, char *class, int type, int
nr_channels, int nr_dim, int dim1, int dim2, int dim3, int dim4, int
dim5)
```

***DESCRIPTION***

"var_object" creates a new object of the desired type and sizes.

"name" is the name of the object.

"class" is the class name of the var_object.

"type" is the type of data stored in the var_object. In principle only the standard C-types are supported, with one exception, the type "PIXEL" is also supported for it is the most often used type.

The following types are supported:

| | |
|---|---|
| PIXEL_T | 1 |
| CHAR_T | 2 |
| SHORT_T | 4 |
| INT_T | 8 |
| LONG_T | 16 |
| FLOAT_T | 32 |
| DOUBLE_T | 64 |

"nr_channels" is the number of channels that each elements consist of. This special dimension is added because in a number of cases it is more convenient. For example, complex data is considered as two-channel to show better compatibility with other data (especially where images are concerned).

"nr_dim" specifies both the number of dimensions of the object, and the number of parameters (of the range dim1 ... dim5) that are valid. All dimensions over "nr_dim" are set to 1 so when calling this function it is allowed and taken into account that not all of the parameter of dim1 .. dim5 are supplied (This is allowed in the C-language). So for example if you say:

```
var_object("my_object","array",FLOAT_T,1,1,128);
```

an object with the name "my_object" and of class "array" is created which is a one-dimensional array of 128 floats.

Even null-dimensional array’s (which are in fact scalars) are allowed, simply specify "nr_dim" as 0 and you have a single variable of the desired type.

The maximum number of dimensions is currently set to 5.

***NOTE***

When using any of the functions for changing the sizes or data-type of the var_object and the system is not capable of allocating enough memory for it, then the contents of the var_object concerned will NOT have been lost. The var_object will be off the same sizes as before with all its data intact.

***RETURN VALUES***

A pointer to the newly defined var_object on success.
NULL pointer on failure

***SEE ALSO***

destroy_var_object var_object_by_name show_var_object_info list_var_objects write_var_object read_var_object var_object_convert var_object_to_image image_to_var_object var_object_copy set_var_object_type set_var_object_size set_var_object_data set_var_object_class set_var_object_comment

---

## *var_object_by_name*

***NAME***

var_object_by_name - get pointer of var_object by its name

***SYNOPSIS***

```
#include "im_proto.h"

VAR_OBJECT *var_object_by_name(char *name, int case_sensitive)
```

***DESCRIPTION***

If the pointer to an var_object is not at hand you obtain that pointer by use of this function. "name" is the name of the var_object, "case_sensitive" specifies whether a distinction between lower case and upper case characters should be made. If it is zero then no distinction is made.

***RETURN VALUES***

Pointer to the requested var_object on success.
NULL pointer if var_object "name" does not exist.

***SEE ALSO***

var_object destroy_var_object show_var_object_info

---

## *var_object_convert*

***NAME***

var_object_convert - convert a var_object into another type

***SYNOPSIS***

```
#include "objectsp.h"

int var_object_convert(VAR_OBJECT *source, VAR_OBJECT *destination,
int out_type)
```

***DESCRIPTION***

Convert a var_object into another data-type. "destination" will become a var_object that is of equal sizes as "source" but of type "out_type". If "out_type" equals zero then "out_type" will be equal to the type of "destination", so the data of "source" will be converted to the type of "destination" and stored in "destination". For a full listing of all available data-types see var_object()

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

var_object  var_object_to_image  image_to_var_object

---

## *var_object_copy*

***NAME***

var_object_copy - copy the contents of one var_object to another

***SYNOPSIS***

```
#include "objectsp.h"

int var_object_copy(VAR_OBJECT *obj1, VAR_OBJECT *obj2)
```

***DESCRIPTION***

Copy the contents of the var_object "obj1" to the var_object "obj2". The type and sizes of the second object are adjusted to match that of the first one.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

var_object

---

## *var_object_ok*

***NAME***

var_object_ok - check if the pointer is a valid var_object pointer

***SYNOPSIS***

```
#include "objectsp.h"

int var_object_ok(VAR_OBJECT *var_object)
```

***DESCRIPTION***

The pointer "var_object" is checked if it points to a valid var_object. The linked list in which all the var_objects are present is scanned for the occurrence of "var_object". If no var_object exist with this pointer, an error is generated and the following message is added to the error-stack:

```
Non existing var_object pointer.
```

The function is_var_object() performs the same check and has the same return values but does not generate an error (and nothing is added to the error-stack).

***RETURN VALUES***

IE_OK (1) if the pointer is a valid var_object.
IE_NOT_OK (0) if "var_object" does not point to a var_object.

***SEE ALSO***

is_var_object

## *var_object_to_image*

***NAME***

var_object_to_image - convert a var_object into an image

***SYNOPSIS***

```
#include "objectsp.h"

int var_object_to_image(VAR_OBJECT *object, IMAGE *image, int
type_of_image)
```

***DESCRIPTION***

Convert the var_object "object" into the image "image". "type_of_image" specifies the type that the image will become. If "type_of_image" is zero then the type of the image itself will be taken. Var_objects with more than 3 dimensions cannot be converted into an image (not counting the nr_channels dimension).

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

var_object  image_to_var_object

## *vi*

## *$*

## *ls*

## *cd*

## *sh*

## *pwd*

## *grep*

***NAME***

vi, $, ls, cd, sh, pwd, grep - execute shell commands

***PLATFORM***

UNIX.

***SYNOPSIS***

```
$<command>

ls <name>

cd <dirname>

sh

pwd

grep <pattern> <filename>
```

***DESCRIPTION***

With "$<command>" a single command is executed by the operating system and control is directly returned to SCIL.  For instance "$date" displays current date and time. Several  frequently used operating system commands can be typed directly which are: "vi", "ls", "sh", "cd", "pwd" and the "grep" command.

***EXAMPLE***

```
[C1] vi filename
[C2] ls *.c
[C3] sh
[C4] cd test
```

The first command will activate the UNIX screen editor concerning the file "filename" and the second will give a directory list of all files with extension ".c". With the command "sh" another shell (/bin/sh) will be initiated to give a sequence of operating system commands before returning control. The "cd" command can be used to change the current working directory, in this case "test"

## *vkuwahara*

***NAME***

vkuwahara - 3D kuwahara filter

***SYNOPSIS***

```
#include "im_proto.h"

int vkuwahara(IMAGE *in, IMAGE *out, int fsize, int vari)
```

***DESCRIPTION***

The vkuwahara() filter operates on a 3*3*3 window in the 3D image "in" and stores the result in image "out". The filter size "fsize" can be set to either 27 or 19 (27 is the entire 3*3*3 window, 19 is the 3*3*3 window without the 8 corner voxels).

The window is divided into 8 octants each containing 8 (27 window) or 7 (19 window) voxels. From each octant the variance or relative variance of the voxels values is determined. The new voxels value will be the mean value of the octant that shows the smallest (relative) variance. For computing speed reasons, the difference between the maximum and the minimum of the voxels in each octant is taken as a measure for the variance.

The parameter "vari" determines whether the variance of a window is to be weighted by the mean (1) or not (0).

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

## *vlaplace*

***NAME***

vlaplace - 3D laplace filter.

***SYNOPSIS***

```
#include "im_proto.h"

int vlaplace(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

Differential edge detection based on the Laplacian operator:

```
     d^2     d^2     d^2
D = ----- + ----- + ----- .
     dx^2    dy^2    dz^2
```

Image "in" is filtered and the result is stored in image "out".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

---

## *vlinear*

***NAME***

vlinear - 3D linear filter

***SYNOPSIS***

```
#include "im_proto.h"

int vlinear(IMAGE *in, IMAGE *filter, IMAGE *out)
```

***DESCRIPTION***

Perform a linear filter on image "in" and store the result in image "out". The weigh-factors are determined by the 3*3*3 image "filter". For each voxel of image "in", the voxels in a 3*3*3 window around this voxel are multiplied by the corresponding weigh-factor from image "filter". The sum of the multiplications, divided by the total sum of the weigh-factors is the new value the voxel in image "out".

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

---

## *vmedian*

***NAME***

vmedian - 3D median filter

***SYNOPSIS***

```
#include "im_proto.h"

int vmedian(IMAGE *in, IMAGE *out, int fsize)
```

***DESCRIPTION***

vmedian() sorts the voxels in the filter window in image "in" on value. The new voxel value in image "out" will be the median value of the sorted voxels. The size of the filter window can be set using "fsize". Valid values are 7 (the voxel plus its direct neighbors on the X-, Y and Z-axis), 19 (a 3*3*3 cube around the voxel without the 8 corner voxels) and 27 (the complete 3*3*3 cube around the voxel). This filter will remove extreme values in the image due to noise, while preserving edges.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

vpercentile

---

## *vpercentile*

***NAME***

vpercentile - 3D percentage filter

***SYNOPSIS***

```
#include "im_proto.h"

int vpercentile(IMAGE *in, IMAGE *out, int perc)
```

***DESCRIPTION***

vpercentile() takes a 27 voxel window (3*3*3) around each voxel in the image "in", sorts them on value. The new voxel-value in the image "out" will be the value that is at the specified position (by "perc") of the sorted row of 27 voxels. So, if 100% is specified, the maximum voxel value in the window will be taken. 0% means the minimum value. Finally, 50% is exactly equal to the median filter on 27 voxels.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

vmedian

---

## *vsobel*

***NAME***

vsobel - 3D Sobel filter.

***SYNOPSIS***

```
#include "im_proto.h"

int vsobel(IMAGE *in, IMAGE *out, int weight_factor)
```

***DESCRIPTION***

Differential edge detection based upon the Sobel operator. The image "in" is filtered and the result is stored in image "out". This filter detects gradients in the image corresponding to edges of the object(s): sudden jumps in voxel values will be recognized and the new voxel value is proportional to the size of the jump. "weight_factor" specifies the distance between two z-slices relatively to the distance between two voxels in the x and y direction. "weight_factor" is a shift factor in bits (0=*1; 1=*2; 2=*4 etc.).

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

---

## *warp_image*

***NAME***

warp_image - adjust one image to the sizes of another

***SYNOPSIS***

```
#include "im_proto.h"

int warp_image(IMAGE *in, IMAGE *out)
```

***DESCRIPTION***

"copy" image "in" to image "out", even when they do not have the same dimensions. The function fblow() is used to do the scaling of the image if the sizes of "in" and "out" are not equal.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

copy_im

---

## *width*

***NAME***

width - width of object

***SYNOPSIS***

```
#include "im_aio.h"

int width(LIST *link)
```

***DESCRIPTION***

| | | |
|---|---|---|
| link | - | Link pointing to the object |

AIO primitive to obtain value of an object feature

width() returns the width of the object pointed to by "link".

This feature need not be specified beforehand as it is automatically measured during the labeling process

***RETURN VALUES***

The width of the object in pixels on succes
0 if link is not an object

***SEE ALSO***

measure  object_shape_meas  object_dens_meas  list_label

## *win_to_comp*

***NAME***

win_to_comp - add rectangular window from image to composite photo

***SYNOPSIS***

```
#include "image.h"
#include "silo.h"

int win_to_comp(COMPTR comptr, IMAGE *image, int left, int top, int
sizex, int sizey)
```

***DESCRIPTION***

| | | |
|---|---|---|
| comptr | - | Pointer to composite photo. |
| image | - | Image which contains the part-image. |
| left | - | Start x-coordinate of part-image. |
| top | - | Start y-coordinate of part-image. |
| sizex | - | Part-image width. |
| sizey | - | Part-image height. |

win_to_comp() copies the part of the image "image" from location ("left","top") and with sizes "sizex"*"sizey" to the composite photo "comptr"

***RETURN VALUES***

The position where the part-image went to:

| | | |
|---|---|---|
| x-start-coordinate | - | function value modulo 2048. |
| y-start-coordinate | - | function value div 2048. |

## *wrap*

***NAME***

wrap - pixel wrap around

***SYNOPSIS***

```
#include "im_proto.h"

int wrap(IMAGE *in, IMAGE *out, int hdispl, int vdispl, int zdispl)
```

***DESCRIPTION***

Wrap around (scroll) the pixels in image "in" with step size "hdispl" in horizontal direction, step size "vdispl" in vertical direction and step size "zdispl" in the depth direction and store the result in image "out". Pixels which are shifted out over the image boundaries are shifted in again at the opposite side.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

## ***write***

***NAME***

write - write on a file

***SYNOPSIS***

```
unsigned int write(int fildes, char *buffer, int nbytes)
```

***DESCRIPTION***

This function is an interface to the standard C function as implemented on the current system. The functionality of this function is:

A file descriptor is a word returned from a successful open(), creat(), dup(), or pipe(2) call.

write() writes data from memory to a file. "buffer" is the address of "nbytes" contiguous bytes which are written to the output file. The number of characters actually written is returned. It should be regarded as an error if this is not the same as requested.

***RETURN VALUES***

Returns -1 on error:

bad descriptor,
buffer address,
count;
physical I/O errors.

***SEE ALSO***

creat  open

## *write_var_object*

***NAME***

write_var_object - write a var_object to a file

***SYNOPSIS***

```
#include "objectsp.h"

int write_var_object(VAR_OBJECT *object, char *filename)
```

***DESCRIPTION***

"write_var_object" write the var_object specified by the pointer "object" to a file. When executed, this function puts two files on disk. A file with the given name and the extension ".voh" and a file with the same name but with the extension ".vod". The file with the extension ".voh" is an ASCII header file which describes the var_object. In the file with the extension ".vod" the actual data resides.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

var_object  read_var_object

## *writefile*

***NAME***

writefile - write an image to file

***SYNOPSIS***

```
#include "im_proto.h"

int writefile(IMAGE *image, char *filename, int fileformat)
```

***DESCRIPTION***

Write the image "image" to the file "filename" using the "fileformat" format. The following formats are supported.

| | |
|---|---|
| ICS_F (1) | ICS format; two files per image are written, the data-file with the extension ".ids" and the header-file with the extension ".ics" |
| TIFF_F (2) | Tiff format; the write function is capable of writing TIFF-files according to the TIFF 6.0 specifications. The file will have the extension ".tif". |
| JPEG_F (3) | JPEG format;the file will have the extension ".jpg". |
| TCL_F (4) | TCL_Image format;the file will have the ".dat" extension. |

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

readfile  ics_writefile  tiff_writefile  tcl_writefile  jpeg_writefile

---

## *writf*

***NAME***

writf

***DESCRIPTION***

This is an old function name, only provided for backward compatibility with TCL_Image routines.

See writefile

---

## *xmax*

***NAME***

xmax - maximum X coordinate of an object

***SYNOPSIS***

```
#include "im_aio.h"

int xmax(LIST *link)
```

***DESCRIPTION***

| | | |
|---|---|---|
| link | - | Link pointing to the object |

AIO primitive to obtain value of an object feature

xmax() returns the maximum X coordinate of the object pointed to by "link".

This feature need not be specified beforehand as it is automatically measured during the labeling process.

***RETURN VALUES***

The maximum X coordinate of the object on success
0 if link is not an object

***SEE ALSO***

measure  object_shape_meas  object_dens_meas  list_label

## *xmin*

***NAME***

xmin - minimum X coordinate of an object

***SYNOPSIS***

```
#include "im_aio.h"

int xmin(LIST *link)
```

***DESCRIPTION***

| | | |
|---|---|---|
| link | - | Link pointing to the object |

AIO primitive to obtain value of an object feature

xmin() returns the minimum X coordinate of the object pointed to by "link".

This feature need not be specified beforehand as it is automatically measured during the labeling process.

***RETURN VALUES***

The minimum X coordinate of the object on success
0 if link is not an object

***SEE ALSO***

measure  object_shape_meas  object_dens_meas  list_label

## *xor_im*

***NAME***

xor_im - bitwise xor of image pixels

***SYNOPSIS***

```
#include "im_proto.h"

int xor_im(IMAGE *in1, IMAGE *in2, IMAGE *out)
```

***DESCRIPTION***

Perform a bitwise XOR operation of each element of "in1" with the corresponding element of "in2" and store the result in "out"

***NOTE***

For more powerful image arithmetic expressions (scaling, adding offsets, etc.), use the function eval() .

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

and_im  or_im  invert_im  shift_im

## ***XYZ_ref_white***

## ***RGB_ref_white***

## ***print_XYZ_ref_white***

***NAME***

XYZ_ref_white - set the reference white in XYZ values

RGB_ref_white - set the reference white in RGB values

print_XYZ_ref_white - print the reference white values in XYZ

***SYNOPSIS***

```
#include "color2dp.h"

void XYZ_ref_white(double Xn, double Yn, double Zn)

void RGB_ref_white(int Rn, int Gn, int Bn)

void print_XYZ_ref_white(void)
```

***DESCRIPTION***

When converting the XYZ color-model to other color-models like L*a*b* and L*u*v*
(the latter is currently not implemented in Image), a reference XYZ triplet is needed. This triplet is commonly named "white point" or "reference white". To set the white-point, the function XYZ_ref_white() can be used. "Xn", "Yn" and "Zn" begin the X, Y and Z value of the "white-point". For convenience the same white point can also be set using the function RGB_ref_white(), specifying the point in RGB values, which are then immediately converted to XYZ using the current conversion method (see set_RGB2XYZ_matrix).

print_XYZ_ref_white() displays the current reference white in X, Y, and Z values.

***RETURN VALUES***

None

***SEE ALSO***

set_RGB2XYZ_matrix

## *ymax*

***NAME***

ymax - maximum Y coordinate of an object

***SYNOPSIS***

```
#include "im_aio.h"

int ymax(LIST *link)
```

***DESCRIPTION***

| | | |
|---|---|---|
| link | - | Link pointing to the object |

AIO primitive to obtain value of an object feature

ymax() returns the maximum Y coordinate of the object pointed to by "link".

This feature need not be specified beforehand as it is automatically measured during the labeling process.

***RETURN VALUES***

The maximum Y coordinate of the object on success
0 if link is not an object

***SEE ALSO***

measure  object_shape_meas  object_dens_meas  list_label

## *ymin*

***NAME***

ymin - minimum Y coordinate of an object

***SYNOPSIS***

```
#include "im_aio.h"

int ymin(LIST *link)
```

***DESCRIPTION***

| | | |
|---|---|---|
| link | - | Link pointing to the object |

AIO primitive to obtain value of an object feature

ymin() returns the minimum Y coordinate of the object pointed to by "link".

This feature need not be specified beforehand as it is automatically measured during the labeling process.

***RETURN VALUES***

The minimum Y coordinate of the object on success
0 if link is not an object

***SEE ALSO***

measure  object_shape_meas  object_dens_meas  list_label

## *z_planes*

***NAME***

z_planes - view planes of a 3D image in a 2d image

***SYNOPSIS***

```
#include "im_proto.h"

int z_planes(IMAGE *in, IMAGE *out, int start, int number, int
border, int value)
```

***DESCRIPTION***

z_planes() shows the selected Z planes of the 3D image "in" in the 2D image "out". Starting at plane number "start", "number" planes are laid out in several rows and columns. If "border" is true (not 0), lines are drawn in the output image "out" to separate the different planes. "value" is the grey-value used for drawing the border-lines.

***RETURN VALUES***

IE_OK (1) on success
Negative error status on failure (see im_error.h)

***SEE ALSO***

sfp  dir_maximum

---

## *zcross*

***NAME***

zcross - calculate zero crossings

***SYNOPSIS***

```
#include "im_proto.h"

int zcross(IMAGE *input, IMAGE *output, double threshold)
```

***DESCRIPTION***

Calculate a bit image that contains a skeleton-like picture, containing the positions where the grey-values cross the value "threshold". If "threshold" is zero, the image gives the positions where the sign of the image changes.

***RETURN VALUES***

IE_OK     (1) on success
IE_NOT_OK (0) on failure

***SEE ALSO***

dist_skelet

---

***NAME***

? - show help information

***SYNOPSIS***

?

***PLATFORM***

UNIX, Macintosh.

***DESCRIPTION***

Direct commands available in SCIL:

| | |
|---|---|
| help <command> | describe command |
| ? <pattern> | show variable/function information |
| load <filename> | load program text |
| run | interpret program |
| chain <filename> | load and run the named program |
| list [start],[end] | show program text |
| more [start],[end] | same as list but in chunks |
| time <command> | time a command |
| logon <logfile> | connect "logfile" to session |
| logoff | disconnect "logfile" |
| macro [-i] [-v] <macfile> | execute macro file |
| hist [start],[end] | show history |
| : <pattern> | recall command |
| expand [1/0] | enable/disable command expanding |
| rmvar | clear old variables |
| $ <os_command> | issue a shell command |
| vi <filename> | invoke screen editor |
| ls [options] | directory list |
| sh | start a shell |
| cd | change current working directory |
| pwd | print working directory |
| grep <pattern> <filenames> | Start UNIX grep |

# *Command syntax in alphabetical order*

$<command> .......... 522
:[number] .......... 231
:[pattern] .......... 231
? <pattern> .......... 228
***void*** _exit(int status) .......... 141
***void*** abort(void) .......... 5
***int*** abs(int i) .......... 6
***int*** abs_im(IMAGE *in, IMAGE *out) .......... 7
***void*** abs_pathname(char *path) .......... 37
***int*** absd_im(IMAGE *in1, IMAGE *in2, IMAGE *out) .......... 8
***double*** acos(double x) .......... 467
***int*** acos_im(IMAGE *in, IMAGE *out) .......... 469
***void*** ActivateMenu(ABSTRACT *top) .......... 311
***int*** add_applic_exposure_func(void (*func)(IMAGE *ip)) .......... 9
***int*** add_applic_win_input_func(void (*func)()) .......... 10
***int*** add_complex(IMAGE *in, double real_part, double imaginary_part, IMAGE *out) .......... 11
***int*** add_float(IMAGE *in, double constant, IMAGE *out) .......... 12
***int*** add_im(IMAGE *in1, IMAGE *in2, IMAGE *out) .......... 13
***int*** add_int(IMAGE *in, int constant, IMAGE *out) .......... 14
***int*** AddImageInfo(IMAGE *im, char *name, void *info, void (*dfunc)(void *)) .......... 15
***void*** AddToMenu(char *name, ABSTRACT *menu, ABSTRACT *parentmenu) .......... 311
***IMAGE **** aim_readfile(char *filename, IMAGE *image, int xpos, int ypos) .......... 16
***int*** aio_label(IMAGE *in, IMAGE *out, int connect) .......... 17
***void*** all_im(char *command, int type) .......... 18
***int*** anchor_skelet(IMAGE *in, IMAGE *mask, IMAGE *out, int iter, int endp, int bound) .......... 19
***int*** and_im(IMAGE *in1, IMAGE *in2, IMAGE *out) .......... 20
***double*** angle(LIST *link) .......... 20
***int*** angle_detection(IMAGE *in, IMAGE *out, double thres) .......... 21
***int*** applic_exposure(int send_events, int skip_when_busy) .......... 22
***int*** applic_win_input(int send_events, int skip_when_busy) .......... 23
***int*** apply_frequency_bank(IMAGE *in, IMAGE *bank, IMAGE *out, int begin, int end) .......... 24
***int*** apply_spatial_bank(IMAGE *in, IMAGE *bank, IMAGE *out, int begin, int end) .......... 24
***int*** arbit_dilation(IMAGE *in, IMAGE *out, IMAGE *se, int bound) .......... 25
***int*** arbit_erosion(IMAGE *in, IMAGE *out, IMAGE *se, int bound) .......... 26
***long*** area(LIST *link) .......... 26
***char **** asctime(struct tm *tp) .......... 27
***double*** asin(double x) .......... 467
***int*** asin_im(IMAGE *in, IMAGE *out) .......... 469
***double*** atan(double x) .......... 467
***int*** atan_im(IMAGE *in, IMAGE *out) .......... 469
***double*** atan2(double x, double y) .......... 467
***int*** atan2_im(IMAGE *in1, IMAGE *in2, IMAGE *out) .......... 469
***int*** atexit(void (*func)(void)) .......... 29
***double*** atof(char *nptr) .......... 30
***int*** atoi(char *nptr) .......... 30

***long*** atol(char *nptr) ........ 30
***int*** auto_display(int mode) ........ 32
***int*** auto_plane(int flag) ........ 33
***int*** auto_point(int flag) ........ 34
***int*** b_to_comp(COMPTR comptr, int sizex, int sizey, PIXEL *buf) ........ 35
***int*** back_project(VAR_OBJECT *input, VAR_OBJECT *data, VAR_OBJECT *output, int width) ........ 36
***int*** bank_frequency_response(IMAGE *bank, IMAGE *out, int begin, int end) ........ 24
***void*** base_name(char *bname, const char *path) ........ 37
***double*** bend(LIST *link) ........ 40
***int*** benke(IMAGE *pat1, IMAGE *pat2, IMAGE *out, int maxiter, double gain, double convergence, int width, int height, int depth) ........ 41
***int*** bernsen_threshold(IMAGE *in, IMAGE *out, int filt_x, int filt_y, int max_diff) ........ 100
***int*** bi_threshold(IMAGE *in, IMAGE *out, int low, int high) ........ 496
***int*** bin_disp_colors(int fg, int bg) ........ 42
***int*** binary_to_grey(IMAGE *in, IMAGE *out, int val) ........ 43
***int*** binary_to_plane(IMAGE *in, IMAGE *out, int plane) ........ 43
***int*** bit_ok(int value) ........ 44
***int*** blow(IMAGE *in, IMAGE *out, int hfact, int vfact, int dfact, int adjust) ........ 45
***double*** box_dimension(IMAGE *input, IMAGE *mask, int fsizemin, int fsizestep, int fsizemax) ........ 46
***void ****bsearch(void *key, void *base, size_t n, size_t size, int (*cmp)(void *key, void *datum)) ........ 48
***int*** buf_from_silo(SILOPTR siloptr, int silo_key, PIXEL *buf) ........ 50
***int*** buf_to_silo(SILOPTR siloptr, int silo_key, PIXEL *buf, int sizex, int sizey) ........ 51
***double*** cabs(struct { double x, y;}z) ........ 241
***int*** calc_greyvalue(IMAGE *image) ........ 52
***int*** calibrated_density(IMAGE *label_im, IMAGE *grey_im, char *fname, int append, VAR_OBJECT *table) ........ 53
***void ****calloc(unsigned int nelem, unsigned int elsize) ........ 314
***int*** canny(IMAGE *in, IMAGE *out, IMAGE *Lx, IMAGE *Ly, double sigma, double acc, int fwidth, int nonmax) ........ 54
cd <dirname> ........ 522
***double*** ceil(double x) ........ 146
chain filename ........ 55
***int*** chaincode_to_image(VAR_OBJECT *input, IMAGE *image) ........ 56
***int*** chaincode_to_xy(VAR_OBJECT *input, VAR_OBJECT *output, int offset) ........ 57
***int*** change_image_size(IMAGE *im, int width, int height, int depth) ........ 58
***int*** channel_bi_threshold(IMAGE *in, IMAGE *out, double min1, double max1, double min2, double max2,double min3, double max3,double min4, double max4) ........ 59
char *sys_errlist[]; ........ 360
***int*** chdir(char *dirname) ........ 60
***int*** check_image_integrity(int print) ........ 61
***void*** check_status(int status, char *str) ........ 62
***int*** clear_im(IMAGE *out) ........ 63
***void*** clear_image_flag(IMAGE *im, int flag) ........ 431
***int*** clear_part_image(IMAGE *im, int sx, int sy, int sz, int width, int height, int depth) ........ 63
***int*** clear_var_object(VAR_OBJECT *object) ........ 64
***void*** clearerr(FILE *stream) ........ 65

***int*** clip(IMAGE *in, IMAGE *out, int lowest, int highest)......66
***long*** clock(void)......27
***int*** close(int fildes)......66
***void*** close_comp(COMPTR comptr)......67
***int*** close_silo(SILOPTR siloptr)......67
***int*** closing3x3(IMAGE *in, IMAGE *out, int iter, int con, int bound)......68
***CLUT ******clut_by_name(char *name, int case_sensitive)......68
***int*** clut_ok(CLUT *clut)......69
***int*** cmp_pixels(IMAGE *in1, IMAGE *in2, VAR_OBJECT *first)......70
***int*** color_get_model_size(int model)......71
***void*** com_dialog(char *string)......72
***void*** compact_silo(SILOPTR siloptr)......73
***int*** complex_im(IMAGE *in1, IMAGE *in2, IMAGE *out)......73
***int*** compute_clut(CLUT *clut)......74
***int*** con_ok(int con)......75
***int*** con6_ok(int con)......76
***int*** conjugate_im(IMAGE *in, IMAGE *out)......76
***int*** conjugate_mul_im(IMAGE *in1, IMAGE *in2, IMAGE *out)......77
***int*** constr_distance(IMAGE *in, IMAGE *constraint, IMAGE *out, int hstep, int dstep, int kstep, int b)......78
***int*** contour(IMAGE *in, IMAGE *out, int edge, int conn, int obj_bkg)......79
***int*** contrast_stretch(IMAGE *in, IMAGE *out, double lperc, double hperc)......79
***int*** convert(IMAGE *in, IMAGE *out, int out_type)......80
***int*** convert_cmodel(IMAGE *in, IMAGE *out, cmodel)......81
***int*** convolution(IMAGE *in, IMAGE *conv, IMAGE *out, int addval, int divval)......82
***int*** cooccur(IMAGE *in, IMAGE *out, int xdist, int ydist)......83
***int*** copy_channel(IMAGE *in, IMAGE *out, int inchan, int outchan)......83
***CLUT ******copy_clut(CLUT *source, CLUT *dest, char *name)......84
***HISTOGRAM ******copy_histogram(HISTOGRAM *srchisto, HISTOGRAM *desthisto)......92
***int*** copy_im(IMAGE *in, IMAGE *out)......84
***int*** copy_object(IMAGE *src_im, IMAGE *dst_im, LIST *link)......85
***int*** copy_part_image(IMAGE *in, BOOL_MASK *mask, IMAGE *out, int sx, int sy, int sz, int width, int height, int depth, int dx, int dy, int dz, int clear)......86
***int*** copy_part_image(IMAGE *in, IMAGE *out, int sx, int sy, int sz, int width, int height, int depth, int dx, int dy, int dz)......86
***double*** cos(double x)......467
***int*** cos_im(IMAGE *in, IMAGE *out)......469
***double*** cosh(double x)......467
***int*** cosh_im(IMAGE *in, IMAGE *out)......469
***double*** covariance(IMAGE *in1, IMAGE *in2)......87
***int*** covmatrix(IMAGE *in, VAR_OBJECT *out, int width, int height)......87
***int*** covplanematrix(IMAGE *in, VAR_OBJECT *out)......88
***double*** cr(LIST *link)......88
***int*** creat(char *name, int mode)......89
***CLUT ******create_clut(int type, char *name)......90
***void*** create_diff_window(IMAGE *image)......95
***int*** create_display(IMAGE *image, int xp, int yp, int xs, int ys)......91
***HISTOGRAM ******create_histogram(char *name, int chans, int dims, int dim1, int dim2, int dim3, int dim4, int dim5)......92

***IMAGE*** *******create_image(char *name, int type, int lenx, int leny, int lenz).............................. 94
***void*** create_live_window(IMAGE *image)............................................................................ 95
***SILOPTR*** create_silo(char *siloname)..................................................................................... 95
***char*** *******ctime(time_t *tp) ........................................................................................................... 27
***int*** cube_view(IMAGE *in, IMAGE *out, int plane, int view, int interaction)....................... 97
***int*** decrement_im(IMAGE *in, IMAGE *out)........................................................................ 98
***void*** default_images(int number)........................................................................................... 99
***int*** defuz(IMAGE *in, IMAGE *out, int filt_x, int filt_y, double thr).................................. 100
***int*** deiconify_window(IMAGE *im).......................................................................................... 393
***int*** del_im_exposure_func(void (*fp)(IMAGE *)) ................................................................ 249
***int*** del_im_input_func(void (*fp)(IMAGE *, int, int, int, IM_EVENT)).............................. 249
***int*** dens_limits(int minimum, int maximum, double max_opt_dens).................................... 102
***int*** density(IMAGE *label_im, IMAGE *grey_im, char *fname, int append)...................... 104
***int*** destroy_clut(CLUT *clut)...................................................................................................... 105
***int*** destroy_histogram(HISTOGRAM *histo)......................................................................... 92
***int*** destroy_image(IMAGE *im)............................................................................................... 105
***int*** destroy_var_object(VAR_OBJECT *obj)............................................................................ 106
***int*** dialog_options(int nfew, int nsome, int nmany, int nhuge, int f_feedb, int s_feedb, int m_feedb, int h_feedb, int g_feedb, int with_arrows, int with_range)................................... 107
***int*** dialog_stay_up(int flag)....................................................................................................... 108
***int*** dialog_wm(int number) ......................................................................................................... 108
***int*** different_ok(int val1, int val2, char *text).......................................................................... 109
***double*** difftime(time_t time2, time_t time1)............................................................................... 27
***int*** dilation3x3(IMAGE *in, IMAGE *out, int iter, int con, int bound) ................................. 110
***int*** dir_maximum(IMAGE *in, IMAGE *out, int dir)............................................................ 111
***void*** dir_name(char *dname, const char *path)......................................................................... 37
***int*** disp_circle(IMAGE *im, int x, int y, int r)........................................................................ 112
***int*** disp_draw_mode(int mode).................................................................................................... 112
***int*** disp_draw_value(int value)..................................................................................................... 112
***int*** disp_oval(IMAGE *im, int x, int y, int xr, int yr).............................................................. 112
***int*** disp_rect(IMAGE *im, int x, int y, int xsize, int ysize)..................................................... 112
***int*** disp_srect(IMAGE *im, int x, int y, int xsize, int ysize).................................................. 112
***int*** disp_text(IMAGE *im, int x, int y, char *str)........................................................................ 112
***int*** disp_text_font(char *font) ..................................................................................................... 113
***int*** disp_vector(IMAGE *im, int x1, int y1, int x2, int y2)....................................................... 113
***int*** display_image(IMAGE *im) ................................................................................................. 114
***double*** dist_average(IMAGE *input, IMAGE *mask, double threshold, int background).... 115
***int*** dist_skelet(IMAGE *in, IMAGE *out, int angle, int hstep, int dstep, int action)............ 116
***int*** distance(IMAGE *in, IMAGE *out, int hstep, int dstep, int kstep, int edge).................. 117
***int*** dither(IMAGE *in, IMAGE *out)........................................................................................ 118
***int*** div_complex(IMAGE *in, double real_part, double imaginary_part, IMAGE *out)........ 11
***int*** div_float(IMAGE *in, double constant, IMAGE *out)........................................................ 12
***int*** div_im(IMAGE *in1, IMAGE *in2, IMAGE *out)............................................................. 13
***int*** div_int(IMAGE *in, int constant, IMAGE *out).................................................................. 14
***int*** do_alert(char *str1, ...).......................................................................................................... 119
***int*** doff(void)................................................................................................................................ 32
***int*** don(void)................................................................................................................................. 32
***int*** draw_line(IMAGE *image, int x1, int y1, int x2, int y2, int value)................................ 120
***int*** drawcurve(VAR_OBJECT *input, IMAGE *output, int value, int smooth, int circ)...... 121

***int*** dump_histogram(HISTOGRAM *histo, char *file, int num) .......... 235
***int*** dump_var_object(VAR_OBJECT *object, char *filename, int number) .......... 122
***int*** dyn_link(char *library, int verbose) .......... 123
***int*** dyn_unlink(char *library, int verbose) .......... 123
***double*** eccentr(LIST *link) .......... 124
***char **** ecvt(double value, int ndigit, int *decpt, int *sign) .......... 125
***double*** edge_average(IMAGE *input, IMAGE *mask, int filtersize, int usegrad) .......... 126
***int*** edge_object(IMAGE *image, LIST  *link) .......... 127
***int*** edge_ok(int bound) .......... 127
***int*** edge_preserve(IMAGE *in1, IMAGE *in2, IMAGE *out, int filtx, int filty) .......... 128
***int*** eigen_segmentation(IMAGE *in, IMAGE *out, int size, int nr, double scale) .......... 129
***int*** eigenfilters(IMAGE *in, IMAGE *out, int width, int height) .......... 130
***int*** eigenvectors(VAR_OBJECT *obj, VAR_OBJECT *vecs, VAR_OBJECT *vals) .......... 131
***int*** entropy_threshold(IMAGE *in, IMAGE *out, double fraction) .......... 131
***int*** equal_images(IMAGE *im1, IMAGE *im2) .......... 132
***int*** equalize(IMAGE *in, IMAGE *out) .......... 133
***int*** equivalent_im(IMAGE *in1, IMAGE *in2, IMAGE *out) .......... 133
***int*** erosion3x3(IMAGE *in, IMAGE *out, int iter, int con, int bound) .......... 134
***void*** err_report(void) .......... 135
***int*** eval(char *expression, int border) .......... 136
***int*** even_ok(int value, char *text) .......... 139
***int*** EventType(IM_EVENT event) .......... 140
***void*** exit(int status) .......... 141
***double*** exp(double x) .......... 142
***int*** exp_im(IMAGE *in, IMAGE *out) .......... 143
***int*** exp10_im(IMAGE *in, IMAGE *out) .......... 144
expand <mode> .......... 145
***double*** fabs(double x) .......... 146
***int*** fast_fourier(IMAGE *in, IMAGE *out, int direction) .......... 147
***int*** fast_hartley(IMAGE *in, IMAGE *out, int norescaling) .......... 148
***int*** fblow(IMAGE *in, IMAGE *out, double hfact, double vfact, double dfact, int adjust)... 149
***int*** fclose(FILE *stream) .......... 150
***char **** fcvt(double value, int ndigit, int *decpt, int *sign) .......... 125
***FILE **** fdopen(int fildes, char *type) .......... 181
***int*** feof(FILE *stream) .......... 65
***int*** ferror(FILE *stream) .......... 65
***int*** fflush(FILE *stream) .......... 150
***int*** fg_buffers(void) .......... 151
***int*** fg_channels(void) .......... 152
***int*** fg_close(void) .......... 153
***int*** fg_depth(void) .......... 154
***int*** fg_exec(const void *input, int ilength, void *output, int olength) .......... 155
***int*** fg_freeze(void) .......... 161
***int*** fg_gain(int *pgain) .......... 156
***int*** fg_get_datasize(void) .......... 157
***const char **** fg_get_last_error(void) .......... 158
***int*** fg_get_rgb_order(void) .......... 159
***int*** fg_getdata(int framenumber, void *storage, int offsetx, int offsety, int incrx, int incry, int countx, int county, int type) .......... 160

***int*** fg_grab(int firstchan, int firstbuf, int nchannels) ..... 161
***int*** fg_grab_image(IMAGE *out, int chan, int width, int height, int slices, int hstep, int vstep, int tstep, int offx, int offy) ..... 162
***int*** fg_grab_next(void) ..... 163
***int*** fg_grab_series(int startchannel, int startbuffer, int nchannels, int nbuffers, int nskip) .... 164
***int*** fg_height(void) ..... 176
***int*** fg_init(const char *initfile) ..... 165
***int*** fg_load(const char *dllname) ..... 166
***int*** fg_maxdepth(void) ..... 154
***int*** fg_maxheight(void) ..... 167
***int*** fg_maxwidth(void) ..... 167
***int*** fg_minheight(void) ..... 168
***int*** fg_minwidth(void) ..... 168
***int*** fg_offset(int *poffset) ..... 169
***int*** fg_set_input_lut(const void *ptable, int nelem) ..... 170
***int*** fg_setdepth(int depth) ..... 171
***int*** fg_setgain(int gain) ..... 156
***int*** fg_setoffset(int offset) ..... 169
***int*** fg_setres(int width, int height) ..... 172
***int*** fg_setroi(int ox, int oy, int ix, int iy, int cx, int cy) ..... 173
***int*** fg_settype(int type) ..... 174
***int*** fg_type(int *ptype) ..... 174
***int*** fg_width(void) ..... 176
***int*** fgetc(FILE *stream) ..... 206
***int*** fgetpos(FILE *stream, fpos_t *ptr) ..... 177
***char ****fgets(char *s, int n, FILE *stream) ..... 209
***int WINAPI*** fgr_buffers(void) ..... 151
***int WINAPI*** fgr_channels(void) ..... 152
***int WINAPI*** fgr_close(void) ..... 153
***int WINAPI*** fgr_depth(void) ..... 154
***int WINAPI*** fgr_exec(const void *input, int ilength, void *output, int olength) ..... 155
***int WINAPI*** fgr_freeze(void) ..... 161
***int WINAPI*** fgr_gain(int *pgain) ..... 156
***int WINAPI*** fgr_get_datasize(void) ..... 157
***int WINAPI*** fgr_get_last_error(char *storage) ..... 158
***int WINAPI*** fgr_get_rgb_order(void) ..... 159
***int WINAPI*** fgr_getdata(int framenumber, void *storage, int offsetx, int offsety, int incrx, int incry, int countx, int county, int type) ..... 160
***int WINAPI*** fgr_grab(int firstchan, int firstbuf, int nchannels) ..... 161
***int WINAPI*** fgr_grab_next(void) ..... 163
***int WINAPI*** fgr_grab_series(int startchan, int startbuf, int nchannels, int nbuffers, int nskip)164
***int WINAPI*** fgr_height(void) ..... 176
***int WINAPI*** fgr_init(const char *initfile) ..... 165
***int WINAPI*** fgr_maxdepth(void) ..... 154
***int WINAPI*** fgr_maxheight(void) ..... 167
***int WINAPI*** fgr_maxwidth(void) ..... 167
***int WINAPI*** fgr_minheight(void) ..... 168
***int WINAPI*** fgr_minwidth(void) ..... 168
***int WINAPI*** fgr_offset(int *poffset) ..... 169

***int WINAPI*** fgr_offset(int offset)....169
***int WINAPI*** fgr_set_input_lut(const void *ptable, int nelem)....170
***int WINAPI*** fgr_setdepth(int depth)....171
***int WINAPI*** fgr_setgain(int gain)....156
***int WINAPI*** fgr_setres(int width, int height)....172
***int WINAPI*** fgr_setroi(int ox, int oy, int ix, int iy, int cx, int cy)....173
***int WINAPI*** fgr_settype(int type)....174
***int WINAPI*** fgr_type(int *ptype)....174
***int WINAPI*** fgr_width(void)....176
***int*** fgreater0_ok(double value, chr *text)....178
***double*** filter_energy_ratio(IMAGE *in1, IMAGE *in2, IMAGE *filter)....179
***int*** flip(IMAGE *in, IMAGE *out, int axis, int angle)....179
***double*** floor(double x)....146
***double*** fmod(double x, double y)....180
***FILE ****fopen(char *filename, char *type)....181
***int*** fpositive_ok(double value, char *text)....182
***int*** fprintf(FILE *stream, char *format, ...)....383
***int*** fputc(int c, FILE *stream)....390
***int*** fputs(char *s, FILE *stream)....391
***int*** fraction_im(IMAGE *in, IMAGE *out)....183
***int*** frange_ok(double value, double vmin, double vmax, char *text)....184
***int*** fread(void *ptr, int size, int nitems, FILE *stream)....185
***void*** free(void *ptr)....314
***FILE ****freopen(char *filename, char *type, FILE *stream)....181
***double*** frexp(double x, int *exp)....180
***int*** fscanf(FILE *stream, char *format, ...)....418
***int*** fseek(FILE *stream, long offset, int ptrname)....186
***int*** fsetpos(FILE *stream, fpos_t *ptr)....177
***int*** fstat(int fildes, struct stat *buf)....482
***long*** ftell(FILE *stream)....186
***int*** funequal0_ok(double value, char *text)....187
***int*** fuz_width(double sigma, int deri, double acc, int maxlen)....188
***int*** fuzzy_derivative(IMAGE *in, IMAGE *out, double sigmax, double sigmay, int derix, int deriy, double accx, double accy, int fwidthx, int fwidthy)....188
***int*** fwrite(void *ptr, int size, int nitems, FILE *stream)....185
***int*** g_copy_object(IMAGE *grey_src, IMAGE *mask_src, IMAGE *dst, LIST  *link)....190
***int*** g_copy_object_to(IMAGE *grey_src, IMAGE *mask_src, IMAGE *dst, LIST  *link, int x, int y)....191
***int*** gabor_bank(IMAGE *out, double radial_bandw, double angular_bandw, int nr)....309
***int*** gauss(IMAGE *in, IMAGE *out, double sigmax, double sigmay, double accx, double accy, int fwidthx, int fwidthy)....192
***int*** gauss_deblur(IMAGE *in, IMAGE *out, double sigma, int order, double accuracy, double factor)....193
***int*** gauss_family(IMAGE *in, IMAGE *out, double sigma, int order, double accuracy, int zero, int even)....194
***char ****gcvt(double value, int ndigit, char *buf)....125
***int*** geo_affine(IMAGE *in, IMAGE *out, double A0, double A1, double A2, double B1, double B2, double B3, int method, int adapt, int border)....195
***int*** geo_rotate(IMAGE *in, IMAGE *out, double angle, int method, int adapt, int border)....195

***int*** geo_warp(IMAGE *in, IMAGE *out)........ 195
***BOOL_MASK* ***get_bool_mask(IMAGE *im)........ 197
***int*** get_disp_mode(void) ........ 32
***int*** get_display_mode(IMAGE *image)........ 198
***int*** get_free_entry(SILOPTR siloptr) ........ 198
***void* ***get_im__error_stack(void) ........ 458
***IMAGE* ***get_image_by_name(char *name, int case_sensitive) ........ 199
***int*** get_image_window_info(IMAGE *im, char *buf)........ 199
***int*** get_pixel(IMAGE *im, int x, int y, [int z,] [int/double *val1, ...])........ 200
***int*** get_pixel_range(IMAGE *image, double minval, double maxval)........ 202
***int*** get_pixel_size(IMAGE *im) ........ 203
***int*** get_sizes(SILOPTR siloptr, int silo_key, int *sizex, int *sizey) ........ 203
***int*** get_slice(IMAGE *im3d, IMAGE *im2d, int orientation, int slice_no)........ 204
***int*** get_super_clut(void) ........ 205
***int*** get_super_histo(void) ........ 205
***int*** get_super_im(void)........ 205
***int*** getc(FILE *stream) ........ 206
***int*** getchar(void)........ 206
***char* ***GETENV(const char *name)........ 207
***void* ***GetImageInfo(IMAGE *im, char *name)........ 15
***char* ***getlogin(void)........ 208
***char* ***gets(char *s)........ 209
***int*** getw(FILE *stream) ........ 206
***double*** glc_asymmetry(IMAGE *input, IMAGE *mask, int vectorx, int vectory)........ 210
***double*** glc_contrast(IMAGE *input, IMAGE *mask, int vectorx, int vectory)........ 210
***double*** glc_entropy(IMAGE *input, IMAGE *mask, int vectorx, int vectory)........ 210
***double*** gld_asymmetry(IMAGE *input, IMAGE *mask, int vectorx, int vectory)........ 212
***double*** gld_contrast(IMAGE *input, IMAGE *mask, int vectorx, int vectory) ........ 212
***double*** gld_entropy(IMAGE *input, IMAGE *mask, int vectorx, int vectory)........ 212
***double*** gld_mean(IMAGE *input, IMAGE *mask, int vectorx, int vectory) ........ 212
***double*** glr_greynonuniformity(IMAGE *input, IMAGE *mask)........ 214
***double*** glr_longrunemphasis(IMAGE *input, IMAGE *mask)........ 214
***double*** glr_nonuniformity(IMAGE *input, IMAGE *mask)........ 214
***double*** glr_percentage(IMAGE *input, IMAGE *mask)........ 214
***double*** glr_shortrunemphasis(IMAGE *input, IMAGE *mask)........ 214
***struct tm* ***gmtime(time_t *tp) ........ 27
***double*** gravx(LIST *link)........ 215
***double*** gravy(LIST *link)........ 216
***int*** greater0_ok(int value, char *text) ........ 217
***int*** greduce(IMAGE *in, IMAGE *out, int nlev, int auto_contr)........ 218
grep <pattern> <filename>........ 522
***iut*** grey_dilation(IMAGE *in, IMAGE *out, int filtx, int filty, int filtz)........ 218
***int*** grey_erosion(IMAGE *in, IMAGE *out, int filtx, int filty, int filtz) ........ 219
***double*** grey_mean(LIST *link)........ 220
***int*** grey_morph_arbit(IMAGE *in, IMAGE *filter, IMAGE *out, int norm, int type)........ 221
***int*** grey_morph_diamond(IMAGE *in, IMAGE *out, int fsize_nw_se, int fsize_ne_sw, int type)........ 221
***int*** grey_morph_ellipse(IMAGE *in, IMAGE *out, int x_axis, int y_axis, double orient, int norm, int type) ........ 221

***int*** grey_morph_hollow_ellipse(IMAGE *in, IMAGE *out, int x_axis, int y_axis, double orient, int conn, int type) ......... 221
***int*** grey_morph_round(IMAGE *in, IMAGE *out, int fsize, int norm, int type) ......... 221
***double*** grey_stdev(LIST *link) ......... 223
***double*** grey_sum(LIST *link) ......... 224
***void*** handle_events(void) ......... 432
***int*** handle_pim(int activate) ......... 225
***int*** have_diff(IMAGE *in, IMAGE *out, int mode) ......... 226
***int*** height(LIST *link) ......... 227
help <pattern> ......... 228
***int*** hide_object(IMAGE *image, LIST  *link) ......... 229
***int*** hide_object_at(IMAGE *image, LIST  *link, int x, int y) ......... 229
***int*** hild_skelet(IMAGE *in, IMAGE *out, int iter, int endp, int bound) ......... 230
hist [start], [end] ......... 231
***int*** hist2d(IMAGE *in1, IMAGE *in2, IMAGE *out, int clip) ......... 232
***int*** histdata(IMAGE *in, VAR_OBJECT *data, int len, int clip) ......... 233
***HISTOGRAM**** histo_data(IMAGE *image, HISTOGRAM *histo, int bins, double minrange, double maxrange) ......... 92
***int*** histogram(IMAGE *in) ......... 233
***HISTOGRAM**** histogram_by_name(char *name, int case_check) ......... 234
***int*** histogram_comment(HISTOGRAM *histo, char *comment) ......... 235
***int*** histogram_ok(HISTOGRAM *histo) ......... 234
***int*** histogram_to_image(HISTOGRAM *histo, IMAGE *image, int out_type) ......... 236
***int*** histogram_to_var_object(HISTOGRAM *histo, VAR_OBJECT *object) ......... 236
***int*** hit_or_miss(IMAGE *in, IMAGE *out, IMAGE *se, int bound) ......... 237
***int*** holt_skelet(IMAGE *in, IMAGE *out, int iter, int bound) ......... 238
***int*** homomorphic(IMAGE *in IMAGE *,out, double low_amplitude, double filt_size) ......... 239
***int*** hull(IMAGE *in, IMAGE *out) ......... 240
***double*** hypot(double x, double y) ......... 241
***int*** Ibenke(IMAGE *filter, IMAGE *out, double gain, double convergence, double sigma, int width, int height) ......... 242
***int*** iconify_window(IMAGE *im) ......... 393
***IMAGE**** ics_readfile(char *filename, IMAGE *image, ICS *ics_header, int xpos, int ypos) 243
***int*** ics_writefile(IMAGE *image, char *filename, ICS *ics_header) ......... 244
***int*** IGreyMap(IMAGE *image) ......... 245
***void*** im_begin_func(const char *fname) ......... 246
***void*** im_clear_errors(void) ......... 458
***void*** im_clear_func_stack(void) ......... 458
***void*** im_debug_stack(int flag) ......... 246
***int*** im_eigenvectors(IMAGE *in, VAR_OBJECT *vecs, VAR_OBJECT *vals) ......... 277
***void*** im_end_func(const char *fname) ......... 246
***int*** im_exposure_func(void (*fp)(IMAGE *), int handle_err) ......... 249
***int*** im_from_silo(SILOPTR siloptr, int silo_key, IMAGE *dstimage) ......... 248
***void*** im_get_func_stack_copy(IM_FUNC_STACK *fstack, int *flevel) ......... 458
***int*** im_get_status(void) ......... 246
***int*** im_input_func(void (*fp)(IMAGE *, int, int, int, IM_EVENT), int handle_err) ......... 249
***int*** im_principle_component(IMAGE *in, VAR_OBJECT *vecs, IMAGE *out, int nr) ......... 277
***int*** im_report_error(const char *fname, int status, const char *message) ......... 246
***IM_OHFUNC*** im_set_output_handler(IM_OHFUNC funcptr) ......... 251

***int*** im_to_silo(SILOPTR siloptr, int silo_key, IMAGE *image)........................................... 252
***int*** im_val_ok(IMAGE *image, int a1 [, int v1, int a2, int v2, ... ])...................................... 253
***void*** im1ps(IMAGE *im, char *title, double xsize, double ysize, int fntsize, int border, int where, char *spec)................................................................................................. 254
***void*** im2ps(IMAGE *im1, char *title1, IMAGE *im2, char *title2, double xsize, double ysize, int fntsize, int border, int where, char *spec)................................................ 254
***void*** im3ps(IMAGE *im1, char *title1, IMAGE *im2, char *title2, IMAGE *im3, char *title3, double xsize, double ysize, int fntsize, int border, int where, char *spec)................ 254
***void*** im4ps(IMAGE *im1, char *title1, IMAGE *im2, char *title2, IMAGE *im3, char *title3, IMAGE *im4, char *title4, double xsize, double ysize, int fntsize, int border, int where, char *spec).................................................................................................. 254
***int*** image_ok(IMAGE *image)..................................................................................... 255
***void*** image_output(int stream, const char *format, ...)........................................... 256
***int*** image_readwrite_ok(IMAGE *image)................................................................. 257
***int*** image_text(IMAGE *out, int x, int y, int val, int boxval, int zoom, char *str)............... 257
***int*** image_to_chaincode(IMAGE *image, VAR_OBJECT *output).................................... 258
***int*** image_to_histogram(IMAGE *image, HISTOGRAM *histo)........................................ 236
***int*** ImageMotionEvents(IMAGE *image, int mode)........................................................ 260
***int*** images_ok(IMAGE *image1, IMAGE *image2)........................................................... 261
***int*** imaginary_im(IMAGE *in, IMAGE *out)....................................................................... 261
***LIST**** *Imeasure(IMAGE *grey, IMAGE *binary, int garb, unsigned long shape, unsigned long dens, int print_it, char *file)........................................................................ 262
***int*** increment_im(IMAGE *in, IMAGE *out)...................................................................... 263
***char**** *index(char *s, char c)............................................................................................ 487
***int*** init_func_overload(void)............................................................................................ 264
***void*** init_scil_image(void)................................................................................................ 264
***void*** init_silo(void)............................................................................................................ 265
***int*** initimage(void)............................................................................................................ 265
int errno;...................................................................................................................... 360
int sys_nerr;................................................................................................................. 360
***void*** interpret(char *str)...................................................................................................... 265
***int*** invert_im(IMAGE *in, IMAGE *out)............................................................................ 266
***int*** Irectangle(IMAGE **im, char *mess, int x, int y, int z, int w, int h, int d)..................... 267
***int*** is_clut(CLUT *clut)......................................................................................................... 69
***int*** is_histogram(HISTOGRAM *histo).................................................................................. 234
***int*** is_image(IMAGE *image)............................................................................................. 268
***int*** is_var_object(VAR_OBJECT *var_object)........................................................................ 269
***int*** isalnum(int c)................................................................................................................ 270
***int*** isalpha(int c)................................................................................................................. 270
***int*** isascii(int c)................................................................................................................... 270
***int*** iscntrl(int c)................................................................................................................... 270
***int*** isdigit(int c)................................................................................................................... 270
***int*** isgraph(int c)................................................................................................................. 270
***int*** islower(int c)................................................................................................................. 270
***int*** IsMouseDown(IM_EVENT mouse_event, int button).................................................. 272
***int*** isodata_threshold(IMAGE *in, IMAGE *out)............................................................... 273
***int*** isprint(int c).................................................................................................................. 270
***int*** ispunct(int c)................................................................................................................. 270
***int*** isspace(int c)................................................................................................................. 270

***int*** isupper(int c)....270
***int*** isxdigit(int c)....270
***int*** iter_ok(int iter)....274
***void*** IThreshold(IMAGE *image)....275
***int*** join_channels(IMAGE *in1, IMAGE *in2, IMAGE *in3, IMAGE *in4, IMAGE *out, int im_type, int subtype)....478
***IMAGE **** jpeg_readfile(char *filename, IMAGE *image, int xpos, int ypos)....276
***int*** jpeg_writefile(IMAGE *image, char *filename)....276
***int*** karhunen_loeve(IMAGE *in, IMAGE *out, int start, int end)....277
***int*** KeyPressed(IM_EVENT event)....278
***int*** kirsch_temp(IMAGE *in, IMAGE *out, IMAGE *direction, int flag)....279
***int*** kuwahara(IMAGE *in, IMAGE *out, int fsize)....280
***int*** kuwahara_round(IMAGE *in, IMAGE *out, int fsize)....281
***int*** label(IMAGE *in, IMAGE *out, int conn)....282
***long*** labs(long n)....6
***int*** laplace(IMAGE *in, IMAGE *out, int mask)....283
***double*** laxis(LIST *link)....284
***double*** ldexp(doublex, int n)....180
***int*** lens(IMAGE *output, int width, int height, char *command)....285
***int*** life(IMAGE *in, IMAGE *out, int iter, int bound)....286
list [start],[end]....287
***int*** list_cluts(void)....288
***int*** list_histograms(void)....235
***LIST **** list_label(IMAGE *in, IMAGE *out, int con, int garb)....289
***int*** list_var_objects(void)....290
***int*** ln_im(IMAGE *in, IMAGE *out)....291
load [filename]....292
***int*** local_contrast(IMAGE *in, IMAGE *out, int radius, int frightmarexp, int fleftmarexp, int fleftmargin, int frightmargin)....293
***int*** local_glc_asymmetry(IMAGE *input, IMAGE *output, int fwidth, int fheight, int vectorx, int vectory)....295
***int*** local_glc_contrast(IMAGE *input, IMAGE *output, int fwidth, int fheight, int vectorx, int vectory)....295
***int*** local_glc_entropy(IMAGE *input, IMAGE *output, int fwidth, int fheight int vectorx, int vectory)....295
***int*** local_gld_asymmetry(IMAGE *input, IMAGE *output, int fwidth, int fheight,int vectorx, int vectory)....297
***int*** local_gld_contrast(IMAGE *input, IMAGE *output, int fwidth, int fheight, int vectorx, int vectory)....297
***int*** local_gld_entropy(IMAGE *input, IMAGE *output, int fwidth, int fheight, int vectorx, int vectory)....297
***int*** local_gld_mean(IMAGE *input, IMAGE *output, int fwidth, int fheight, int vectorx, int vectory)....297
***double*** local_glr_greynonuniformity(IMAGE *input, IMAGE *output, int fwidth, int fheight)299
***double*** local_glr_longrunemphasis(IMAGE *input, IMAGE *output, int fwidth, int fheight)299
***double*** local_glr_nonuniformity(IMAGE *input, IMAGE *output, int fwidth, int fheight)..299
***double*** local_glr_percentage(IMAGE *input, IMAGE *output, int fwidth, int fheight)........299
***double*** local_glr_shortrunemphasis(IMAGE *input, IMAGE *output, int fwidth, int fheight)299
***struct tm **** localtime(time_t *tp)....27

***double*** log(double x) ... 142
***double*** log10(double x) ... 142
***int*** log10_im(IMAGE *in, IMAGE *out) ... 300
logoff ... 301
logon <logfile> ... 301
***int*** lookup(IMAGE *in, IMAGE *out, VAR_OBJECT *table, int clip) ... 302
***int*** lower_gskeleton(IMAGE *in, IMAGE *g_out, int border, int endpixel) ... 303
***int*** lower_window(IMAGE *im) ... 393
***int*** lowest_int(IMAGE *in, IMAGE *out) ... 304
ls <name> ... 522
***long*** lseek(int fildes, long offset, int whence) ... 305
macro [-i] [-v] <macrofile> ... 306
***int*** majority(IMAGE *in, IMAGE *out, int bound, int weight) ... 307
***int*** make_color_im(IMAGE *im1, IMAGE *im2, IMAGE *im3, IMAGE *out) ... 308
***int*** make_complex_im(IMAGE *in1, IMAGE *in2, IMAGE *out) ... 73
***int*** make_gabor(IMAGE *out, double fcentral, double sigma_u, double sigma_v, double orientation) ... 309
***IMAGE **** make_image(char *name, int type, int lenx, int leny, int lenz, int posx, int posy) 310
***ABSTRACT **** MakeNewMenu(char *name, ABSTRACT *parentmenu) ... 311
***void **** malloc(unsigned int size) ... 314
***int*** max_element(IMAGE *in, VAR_OBJECT *result, int whole, int dimension) ... 315
***int*** maximum_cost_path(IMAGE *input, VAR_OBJECT *output, int markov, int circular) 316
***int*** maximum_im(IMAGE *in1, IMAGE *in2, IMAGE *out) ... 317
***int*** maximum_trace(IMAGE *input, IMAGE *output, int startx, int starty, int dir, int avglen, int length, int minedge, double minval, VAR_OBJECT *table) ... 318
***LIST **** measure(IMAGE *grey, IMAGE *binary, int garb, int inter, unsigned long shape, unsigned long dens, int print_it, char *file) ... 320
***void **** memchr(void *s, int c, int n) ... 321
***int*** memcmp(void *s, void *t, int n) ... 321
***void **** memcpy(void *s, void *t, int n) ... 321
***void **** memmove(void *s, void *t, int n) ... 321
***void **** memset(void *s, int c, int n) ... 321
***int*** merge(IMAGE *in, IMAGE *out, int direct, int iter) ... 322
***int*** merge_horizontal(IMAGE *in, IMAGE *out, int iter) ... 322
***int*** merge_vertical(IMAGE *in, IMAGE *out, int iter) ... 323
***int*** message_line_info(int code, char *buf) ... 324
***int*** min_element(IMAGE *in, VAR_OBJECT *result, int whole, int dimension) ... 315
***int*** minimum_im(IMAGE *in1, IMAGE *in2, IMAGE *out) ... 325
***int*** mirror(IMAGE *in, IMAGE *out, int direct) ... 326
***int*** mirror_horizontal(IMAGE *in, IMAGE *out) ... 327
***int*** mirror_vertical(IMAGE *in, IMAGE *out) ... 327
***int*** mix(IMAGE *in, IMAGE *out, long thres, long val1, long val2, long val3) ... 328
***int*** mix_filter(IMAGE *in1, IMAGE *in2, IMAGE *sum, IMAGE *diff, IMAGE *alpha) 329
***time_t*** mktime(struct tm *tp) ... 27
***unsigned long*** mmops3x3(IMAGE *X, IMAGE *Y, IMAGE *M, int S, int T, int opcode, int edge) ... 330
***int*** modal_input(buf0, …) ... 334
***double*** modf(double x, double *ip) ... 180
***int*** modulo_im(IMAGE *in1, IMAGE *in2, IMAGE *out) ... 335

more [start],[end] ....287
***int*** MouseMove(IM_EVENT mouse_event) ....336
***int*** MousePress(IM_EVENT mouse_event) ....337
***int*** MouseRelease(IM_EVENT mouse_event) ....338
***int*** mul_complex(IMAGE *in, double real_part, double imaginary_part, IMAGE *out)....11
***int*** mul_float(IMAGE *in, double constant, IMAGE *out)....12
***int*** mul_im(IMAGE *in1, IMAGE *in2, IMAGE *out)....13
***int*** mul_int(IMAGE *in, int constant, IMAGE *out) ....14
***int*** natural_window_size(IMAGE im) ....339
***int*** nearest_int(IMAGE *in, IMAGE *out)....340
***int*** negation_im(IMAGE *in, IMAGE *out) ....340
***int*** next_plane(IMAGE *im, int num) ....341
***LIST ****object_contour(IMAGE *mask, LIST  *link) ....341
***int*** object_dens_meas(IMAGE *grey, IMAGE *mask, LIST  *link, unsigned long bitmap).342
***int*** object_freeman_meas(IMAGE *mask, LIST  *link, unsigned long bitmap)....344
***int*** object_moment_meas(IMAGE *mask, LIST  *link, unsigned long bitmap)....345
***int*** object_rect_to_silo(SILOPTR siloptr, int silo_key, IMAGE *srcimage, LIST *link)....346
***int*** object_shape_meas(IMAGE *mask, LIST  *link, unsigned long bitmap)....347
***int*** objectsize(IMAGE *in, IMAGE *out) ....348
***double*** od_mean(LIST *link)....220
***double*** od_stdev(LIST *link)....223
***double*** od_sum(LIST *link)....224
***int*** odd_fsizes_ok(int fx, int fy, int fmax) ....349
***int*** odd_ok(int value, char *text)....350
***int*** open(char *name, int mode) ....351
***SILOPTR*** open_silo(char *siloname)....351
***int*** opening3x3(IMAGE *in, IMAGE *out, int iter, int con, int bound)....352
***int*** or_im(IMAGE *in1, IMAGE *in2, IMAGE *out)....352
***FPI*** overload_func(char *name, IMAGE *im)....353
***FPI*** overloadable_func(char *name, IMAGE *im) ....353
***int*** palette2color(IMAGE *in, IMAGE *out)....354
***int*** parabolic_closing(IMAGE *in, IMAGE *out, double rho)....355
***int*** parabolic_dilation(IMAGE *in, IMAGE *out, double rho) ....355
***int*** parabolic_erosion(IMAGE *in, IMAGE *out, double rho)....355
***int*** parabolic_opening(IMAGE *in, IMAGE *out, double rho) ....355
***int*** part_from_silo(SILOPTR siloptr, int silo_key, IMAGE *dstimage, int left, int top)....357
***int*** part_image_display(IMAGE *im, int sx, int sy, int sz, int width, int height, int depth)...357
***int*** part_to_silo(SILOPTR siloptr, int silo_key, IMAGE *srcimage, int left, int top, int sizex, int sizey) ....358
***int*** perc_to_pixel(IMAGE *in, int top, int mode, int slicenr, double perc) ....461
***int*** percentile(IMAGE *in, IMAGE *out, int fx, int fy, int num) ....358
***double*** peri(LIST *link) ....359
***void*** perror(char *s) ....360
***int*** phase_im(IMAGE *in, IMAGE *out) ....361
***double*** pix_abs_sum(IMAGE *in)....361
***long*** pix_count(IMAGE *in, int val) ....362
***int*** pix_maxval(IMAGE *in, VAR_OBJECT *result, int whole, int dimension)....363
***int*** pix_minval(IMAGE *in, VAR_OBJECT *result, int whole, int dimension) ....363
***int*** pix_sum(IMAGE *in, VAR_OBJECT *sum)....364

***char ****pix_value_str(IMAGE *im, int x, int y, int z) ............ 364
***PIXEL*** pixval(LIST *link)............ 365
***int*** pl_io_ok(int in, int out)............ 366
***int*** plane_ok(int plane) ............ 367
***int*** plane_to_binary(IMAGE *in, int plane, IMAGE *out)............ 367
***int*** planecopy(IMAGE *in, int inplane, IMAGE *out, int outplane)............ 368
***int*** plot_histogram(IMAGE *in, int action, int clip)............ 368
***int*** point_im(IMAGE **imptr, int *xptr, int *yptr, int *butptr)............ 369
***int*** point_im_display_buf(char *buf, int follow)............ 370
***LIST ****point_object(IMAGE *image, LIST *list)............ 371
***int*** poll_mouse(IMAGEimage.h *image, int *im_x, int *im_y, int *win_x, int *win_y) ..... 372
***int*** positive_ok(int value, char *text) ............ 373
***int*** post_op(IMAGE *out)............ 374
***double*** pow(double x, double y)............ 142
***int*** power_im(IMAGE *in1, IMAGE *in2, IMAGE *out)............ 374
***int*** power_of_2_ok(int value, char *text)............ 375
***int*** pre_op(IMAGE *first, IMAGE *second, int mode, int first_spec, unsigned long sec_type)376
***int*** prefered_model(IMAGE *image, int inout, int model)............ 71
***int*** prewitt_diff(IMAGE *in, IMAGE *out, int mode) ............ 380
***int*** prewitt_temp(IMAGE *in, IMAGE *out, IMAGE *direction, int flag)............ 381
***void*** print_RGB_matrices(void)............ 436
***void*** print_XYZ_ref_white(void) ............ 535
***int*** printf(char *format, ...) ............ 383
***int*** propagation(IMAGE *in, IMAGE *mask, IMAGE *out, int iter, int conn, int edge)...... 385
***int*** ps_head(char *filename, int papersize)............ 386
***int*** ps_image(IMAGE *image, int orient, int unit, double xpos, double ypos, double xsiz, double ysiz, int border, char *comment, int textsize) ............ 386
***int*** ps_tail(void)............ 386
***int*** pseudo(IMAGE *in, IMAGE *out)............ 388
***int*** psremoval(IMAGE *in, IMAGE *out, int bound)............ 388
***int*** put_pixel(IMAGE *im, int x, int y, [int z,] int/doule val1, [int/double val2, ...]) ............ 200
***int*** put_slice(IMAGE *im2d, IMAGE *im3d, int orientation, int slice_no)............ 204
***int*** put_xy_into_image(VAR_OBJECT *input, IMAGE *output, int value) ............ 389
***int*** putc(int c, FILE *stream)............ 390
***int*** putchar(int c)............ 390
***int*** puts(char *s)............ 391
***int*** putw(int w, FILE *stream)............ 390
pwd ............ 522
***int*** qpix(IMAGE *in, char *fname, int zoom, int append)............ 392
***void*** qsort(void *base, size_t n, size_t size, int (*cmp)(void *, void *))............ 48
***void*** quit(void) ............ 392
***int*** raise_window(IMAGE *im) ............ 393
***int*** rand(void)............ 394
***int*** random_filter(IMAGE *out, double mean, double max, int symmetric) ............ 394
***int*** random_im(IMAGE *im, int alt)............ 395
***int*** range_ok(int value, int vmin, int vmax, char *text)............ 396
***int*** raster(IMAGE *in, IMAGE *out, int factor, int ratio)............ 397
***int*** read(int fildes, char *buffer, int nbytes)............ 398
***int*** read_var_object(char *filename, VAR_OBJECT *object) ............ 399

***IMAGE ****readfile(char *filename, IMAGE *image, int xpos, int ypos)................................400
***int*** real_im(IMAGE *in, IMAGE *out)....................................................................401
***int*** real_time_recognizer(IMAGE *in, IMAGE *out, IMAGE *se, int thr, int bound)..........401
***void ****realloc(void *ptr, unsigned int size)..........................................................314
***int*** reduce(IMAGE *in, IMAGE *out, int hfact, int vfact, int dfact, int adjust).....................402
***FILE ****rel_searchfile(char *name, char *envvar, char *pathret)...................................421
***void*** remark()..................................................................................................403
***int*** remove(char *filename)...............................................................................404
***int*** remove_holes(IMAGE *in, IMAGE *out).........................................................404
***int*** RemoveImageInfo(IMAGE *im, char *name)......................................................15
***int*** rename(char *oldname, char *newname).........................................................404
***int*** resample_perp(IMAGE *input, VAR_OBJECT *xy, IMAGE *out, int width, int fitlength, VAR_OBJECT *data, int threshold).........................................................................405
***LIST ****retrieve_object_list(void)........................................................................406
***void*** rewind(FILE *stream)................................................................................186
***int*** RGB_clear_extra(IMAGE *image)......................................................................52
***int*** RGB_gamma_correction(IMAGE *in, IMAGE *out, double r_gamma, double g_gamma, double b_gamma)................................................................................................406
***void*** RGB_ref_white(int Rn, int Gn, int Bn).............................................................535
***int*** rhull(IMAGE *in, IMAGE *out, int dist)...........................................................407
***char ****rindex(char *s, char c)...........................................................................487
***LIST ****rm_list(LIST *list)..................................................................................407
***void*** rm_object(LIST *link)................................................................................408
***int*** rm_silo(SILOPTR siloptr).............................................................................409
***int*** rm_silo_object(SILOPTR siloptr, int silo_key).................................................409
rmvar.......................................................................................................................410
***int*** roberts_diff(IMAGE *in, IMAGE *out, int fsize, int mode).....................................410
***int*** robinson_temp(IMAGE *in, IMAGE *out, IMAGE *direction, int flag).........................412
***IMAGE ****roi_define(char *name, IMAGE *parent, int sx, int sy, int sz, int width, int height, int depth, BOOL_MASK *mask).........................................................................413
***int*** rotate(IMAGE *in, IMAGE *out, int iter).........................................................414
run...........................................................................................................................414
***LIST ****S_Append(INFO *item, LIST *list)..............................................................415
***LIST ****S_BreakList(LIST *list)..........................................................................415
***LIST ****S_CloseList(LIST *list)..........................................................................415
***LIST ****S_CopyList(LIST *list)...........................................................................415
***LIST ****S_Delete(LIST *item, FREEFUNC FreeInfo)....................................................415
***LIST ****S_FindItem(INFO *item, LIST *list)............................................................415
***int*** S_FreeList(LIST *list, FREEFUNC FreeInfo)......................................................415
***LIST ****S_Insert(INFO *item, LIST *list)...............................................................415
***int*** S_Length(LIST *list)....................................................................................415
***LIST ****S_Prefix(INFO *item, LIST *list)................................................................415
***LIST ****S_SortList(LIST *list, COMPAREFUNC compare)..............................................415
***double*** saxis(LIST *link).....................................................................................417
***int*** scanf(char *format, ...)..................................................................................418
***FILE ****searchfile(char *name, char *envvar, char *pathret).........................................421
***int*** set_aio_disp(int mode)..................................................................................422
***int*** set_border(IMAGE *out, double value, int top, int right, int bot, int left, int z_min, int z_max)....................................................................................................422

***int*** set_clut(IMAGE *image, CLUT *clut, int disp) ... 423
***int*** set_color(IMAGE *image, int red_val, int green_val, int blue_val) ... 423
***int*** set_color_model(IMAGE *image, int inout, int model) ... 71
***int*** set_color_model(IMAGE *image, int model) ... 71
***void*** set_common_line(COMMON_LINE *com_line, int type, void *data, int x, int y, int z, int t, int chan, double h_min, double h_max) ... 424
***void*** set_comp_margin(int size) ... 425
***int*** set_complex(IMAGE *im, double real_part, double imaginary_part) ... 425
***void*** set_cross_dim(int cross_x, int cross_y, int cross_z) ... 376
***int*** set_dialog_pos(int x, int y) ... 426
***int*** set_display_mode(IMAGE *image, int mode, int global, int direct_display) ... 427
***int*** set_display_slice(IMAGE *image, int slice) ... 428
***int*** set_dither_mode(IMAGE *image, int mode, int direct_display) ... 429
***int*** set_float(IMAGE *im, double constant) ... 430
***int*** set_im_type(IMAGE *im, int type) ... 430
***void*** set_image_flag(IMAGE *im, int flag) ... 431
***int*** set_image_interaction(int mode) ... 432
***int*** set_int(IMAGE *im, int constant) ... 433
***int*** set_jpeg_quality(int percentage) ... 276
***void*** set_line_editor(int mode) ... 434
***int*** set_menu_pos(int x, int y) ... 435
***int*** set_rgb_bits(int r_bit, int g_bit, int b_bit) ... 435
***void*** set_RGB2XYZ_matrix(int matrix_type) ... 436
***void*** set_RGB2XYZ_mvalues(double m11, double m12, double m13, double m21, double m22, double m23, double m31, double m32, double m33) ... 436
***int*** set_roi_mask(IMAGE *roi_im, BOOL_MASK *mask) ... 438
***int*** set_roi_parent(IMAGE *roi_im, IMAGE *parent) ... 438
***int*** set_roi_pos(IMAGE *roi_im, int sx, int sy, int sz) ... 438
***int*** set_screen_gamma(double gamma) ... 439
***int*** set_sigmoid_shape(float slope, float bending_point) ... 440
***void*** set_start_pos(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4) ... 441
***void*** set_start_sizes(int w1, int h1, int w2, int h2, int w3, int h3, int w4, int h4) ... 441
***void*** set_tiff_compression(int enable) ... 442
***void*** set_tiff_image_number(int number) ... 442
***int*** set_var_object_class(VAR_OBJECT *obj, char *class) ... 443
***int*** set_var_object_comment(VAR_OBJECT *obj, char *comment) ... 443
***int*** set_var_object_data(VAR_OBJECT *obj, int type, int nr_channels, int nr_dim, int dim1, int dim2, int dim3, int dim4, int dim5) ... 444
***int*** set_var_object_size(VAR_OBJECT *obj, int nr_channels, int nr_dim, int dim1, int dim2, int dim3, int dim4, int dim5) ... 445
***int*** set_var_object_type(VAR_OBJECT *obj, int type_of_data) ... 446
***int*** set_window_pos(IMAGE im, int x, int y) ... 447
***int*** set_window_size(IMAGE *im, int sizex, int sizey) ... 447
***void*** setbuffer(FILE *stream, char *buf, int size) ... 448
***void*** SETENV(const char *string) ... 207
***void*** setlinebuf(FILE *stream) ... 448
***void*** SetMacFileCreator(long creator) ... 450
***void*** SetMacFileType(long type) ... 450
***void*** setprompt(char *prompt) ... 451

***int*** setvbuf(FILE *stream, char *buf, int type, int size)........................................................448
***int*** sfp(IMAGE *In, IMAGE *Out, int Orientation, int Background, int Light, int View, int Excitation, int Emission, int Extra_light)...............................................................................452
sh ......................................................................................................................................522
***int*** shape(IMAGE *label_im, VAR_OBJECT *label, VAR_OBJECT *xcentre, VAR_OBJECT *ycentre, VAR_OBJECT *area, VAR_OBJECT *peri, VAR_OBJECT *p2a, VAR_OBJECT *ccoun, VAR_OBJECT *count, int nosmooth)..........................................454
***int*** shift_im(IMAGE *in, IMAGE *out, int nshift)..............................................................455
***void*** show_cur_dir(int flag)....................................................................................................456
***int*** show_dmode_flags(IMAGE *image)................................................................................456
***void*** show_error_stack(void)...................................................................................................458
***int*** show_func_overload(char *spec_func, int im_type, char *file_name)............................457
***void*** show_func_stack(void) ...................................................................................................458
***int*** show_histogram_info(HISTOGRAM *histo) ...................................................................235
***void*** show_image_info(IMAGE *im) .........................................................................................459
***int*** show_menu_layout(char *filename, int show_items) .......................................................460
***int*** show_statistics(IMAGE *in, int mode, int slicenr, int lo_limit, int hi_limit) ..................461
***int*** show_var_object_info(VAR_OBJECT *obj)....................................................................463
***int*** sigma(IMAGE *in, IMAGE *out, int fsize).....................................................................463
***int*** sign_im(IMAGE *in, IMAGE *out) ...................................................................................464
***void*** silo_list(SILOPTR siloptr).................................................................................................465
***int*** silo_to_comp(SILOPTR siloptr, COMPTR comptr, int startlabel, int endlabel).............466
***double*** sin(double x) ....................................................................................................................467
***int*** sin_im(IMAGE *in, IMAGE *out) ......................................................................................469
***int*** single_pixels(IMAGE *in, IMAGE *out, int bound, int conn, int obj_bkg, int detect_rem)470
***double*** sinh(double x) ..................................................................................................................467
***int*** sinh_im(IMAGE *in, IMAGE *out) ....................................................................................469
***int*** skelpoints(IMAGE *in, IMAGE *out, int bound, int opcode, int type)...........................471
***unsigned*** sleep(unsigned seconds)..............................................................................................472
***int*** small_object_removal(IMAGE *in, IMAGE *out, int size) ..............................................472
***int*** sobel_diff(IMAGE *in, IMAGE *out, int mode)..............................................................473
***void*** spb_publish(void *obj, int mess, void *data)...................................................................474
***void*** spb_subscribe(void *obj, void *id, SPBFUNC subscr, void *cldata) ...........................475
***void*** spb_unsubscribe(void *obj, void *id, SPBFUNC subscr) ..............................................475
***int*** spix(IMAGE *in, IMAGE *out)...........................................................................................476
***int*** split(IMAGE *in, IMAGE *out, int direct, int iter)..........................................................477
***int*** split_channels(IMAGE *in, IMAGE *out1, IMAGE *out2, IMAGE *out3, IMAGE *out4) ...........................................................................................................................................478
***int*** split_color_im(IMAGE *in, IMAGE *out1, IMAGE *out2, IMAGE *out3)..................308
***int*** split_horizontal(IMAGE *in, IMAGE *out, int iter).........................................................479
***int*** split_vertical(IMAGE *in, IMAGE *out, int iter).............................................................479
***int*** sprintf(char *s, char *format, ...) ..........................................................................................383
***double*** sqrt(double x) ..................................................................................................................142
***int*** sqrt_im(IMAGE *in, IMAGE *out) .......................................................................................480
***void*** srand(unsigned int seed)...................................................................................................394
***void*** srandom_im(long seed) .......................................................................................................395
***int*** sscanf(char *s, char *format, ...)..........................................................................................418
***int*** standard_gabor(IMAGE *out, double radial_bandw, double fcentral, double angular_bandw, double orientation)..........................................................................................309

***COMPTR*** start_comp(IMAGE *im)........................................ 481
***int*** stat(char *name, struct stat *buf)........................................ 482
***int*** stereo_view(IMAGE *in, int mode, int view, IMAGE *left, IMAGE *middle, IMAGE *out) ........................................ 484
***int*** stop_lens(void)........................................ 285
***int*** strcasecmp(const char *str1, const char *str2)........................................ 485
***char ****strcat(char *s1, char *s2) ........................................ 486
***char ****strchr(char *s, char c)........................................ 486
***int*** strcmp(char *s1, char *s2) ........................................ 486
***char ****strcpy(char *s1, char *s2) ........................................ 486
***size_t*** strcspn(char *s, char *t) ........................................ 487
***char ****strerror(int errnum) ........................................ 487
***size_t*** strftime(char *s, size_t smax, char *fmt, struct tm *tp)........................................ 27
***unsigned int*** strlen(char *s) ........................................ 486
***int*** strncasecmp(const char *str1, const char *str2, size_t num)........................................ 485
***char ****strncat(char *s1, char *s2, int n)........................................ 486
***int*** strncmp(char *s1, char *s2, int n)........................................ 486
***char ****strncpy(char *s1, char *s2, int n)........................................ 486
***char ****strpbrk(char *s, char *t)........................................ 487
***char ****strrchr(char *s, char c) ........................................ 487
***char ****strsave(const char *str) ........................................ 485
***size_t*** strspn(char *s, char *t) ........................................ 487
***char ****strstr(char *s, char *t) ........................................ 487
***double*** strtod(char *s, char **endp)........................................ 30
***char ****strtok(char *s, char *t) ........................................ 487
***long*** strtol(char *s, char **endp, int base) ........................................ 30
***unsigned long*** strtoul(char *s, char **endp, int base) ........................................ 30
***int*** sub_complex(IMAGE *in, double real_part, double imaginary_part, IMAGE *out)........ 11
***int*** sub_float(IMAGE *in, double constant, IMAGE *out) ........................................ 12
***int*** sub_im(IMAGE *in1, IMAGE *in2, out)........................................ 13
***int*** sub_int(IMAGE *in, int constant, IMAGE *out)........................................ 14
***void*** swab(char *from, char *to, int nbytes)........................................ 488
***int*** sync_display(void)........................................ 489
***int*** system(char *string)........................................ 489
***int*** t_morphology(IMAGE *in, IMAGE *out, IMAGE *se, int thr, int bound).................... 490
***int*** tan_im(IMAGE *in, IMAGE *out) ........................................ 469
***double*** tanh(double x) ........................................ 467
***int*** tanh_im(IMAGE *in, IMAGE *out) ........................................ 469
***int*** taylor_expansion(IMAGE *in, IMAGE *out, double sigma, int order, double accuracy, double delta_x, double delta_y, double t) ........................................ 492
***int*** taylor_polynomial(VAR_OBJECT *out, int order, double delta_x, double delta_y, double delta_s) ........................................ 492
***int*** taylor_segmentation(IMAGE *in, IMAGE *out, double dscale, int order, double smooth)493
***IMAGE ****tcl_readfile(char *filename, IMAGE *image, int xpos, int ypos)......................... 494
***int*** tcl_writefile(IMAGE *image, char *filename)........................................ 495
***long*** tell(int fildes)........................................ 305
***int*** threshold(IMAGE *in, IMAGE *out, int level)........................................ 496
***int*** ti_block(IMAGE *out, int block_size, int forgr, int backgr)........................................ 497
***int*** ti_circle(IMAGE *out, int dist, int rad, int line_w, int forg, int back)............................ 497

***int*** ti_fractal(IMAGE *out, double dim)....498
***int*** ti_hlines(IMAGE *out, int dist, int start_p, int line_w, int forg, int back)....499
***int*** ti_ifr(IMAGE *out, double slope)....498
***int*** ti_lines(IMAGE *out, int dist, int start_p, int line_w, int forg, int back)....500
***int*** ti_lshading(IMAGE *out, double top_r, double top_l, double bot_l)....500
***int*** ti_points(IMAGE *out, int dist, int start_p, int width, int forg, int back)....501
***int*** ti_qshading(IMAGE *out, int cval, int top_l)....501
***int*** ti_vlines(IMAGE *out, int dist, int start_p, int line_w, int forg, int back)....502
***IMAGE ****tiff_readfile(char *filename, IMAGE *image, int xpos, int ypos)....503
***int*** tiff_writefile(IMAGE *image, char *filename)....504
time <command>....506
***time_t*** time(time_t *tp)....27
***FILE ****tmpfile(void)....507
***char ****tmpnam(char s[L_tmpnam])....507
***int*** tolower(int c)....508
***int*** toupper(int c)....508
***double*** trans_mean(LIST *link)....220
***double*** trans_stdev(LIST *link)....223
***int*** tri_state_threshold(IMAGE *in, IMAGE *out, int thresh, int val1, int fl1, int val2, int fl2, int val3, int fl3)....509
***int*** truncate_im(IMAGE *in, IMAGE *out)....510
***int*** TWAcquire(IMAGE *image)....511
***int*** TWAcquireArea(IMAGE *image, double res, double xleft, double ytop, double xright, ybottom, int type)....511
***FPI*** type_overload_func(char *name, int type)....353
***FPI*** type_overloadable_func(char *name, int type)....353
***int*** unequal0_ok(int value, char *text)....512
***int*** ungetc(char *c, FILE *stream)....513
***int*** uniform(IMAGE *in, IMAGE *out, int filtx, int filty, int filtz)....514
***int*** uniform_round(IMAGE *in, IMAGE *out, int fsize, int norm)....514
***LIST ****update(LIST *list)....515
***int*** upper_gskeleton(IMAGE *in, IMAGE *g_out, IMAGE *b_out, int metric, int border, int endpixel)....303
***int*** val_check(IMAGE *image, int axis, int value)....516
***VAR_OBJECT ****var_object(char *name, char *class, int type, int nr_channels, int nr_dim, int dim1, int dim2, int dim3, int dim4, int dim5)....517
***VAR_OBJECT ****var_object_by_name(char *name, int case_sensitive)....518
***int*** var_object_convert(VAR_OBJECT *source, VAR_OBJECT *destination, int out_type)519
***int*** var_object_copy(VAR_OBJECT *obj1, VAR_OBJECT *obj2)....519
***int*** var_object_ok(VAR_OBJECT *var_object)....520
***int*** var_object_to_image(VAR_OBJECT *object, IMAGE *image, int type_of_image)......521
***int*** vfuzzy_derivative(IMAGE *in, IMAGE *out, double sigmax, double sigmay, double sigmaz, int derix, int deriy, int deriz, double accx, double accy, double accz, int fwidthx, int fwidthy, int fwidthz)....188
***int*** vgauss(IMAGE *in, IMAGE *out, double sigmax, double sigmay, double sigmaz, double accx, double accy, double accz, int fwidthx, int fwidthy, int fwidthz)....192
***int*** vkuwahara(IMAGE *in, IMAGE *out, int fsize, int vari)....523
***int*** vlaplace(in, out)....524
***int*** vlinear(IMAGE *in, IMAGE *filter, IMAGE *out)....524

***int*** vmedian(IMAGE *in, IMAGE *out, int fsize) ........ 525
***int*** vpercentile(IMAGE *in, IMAGE *out, int perc) ........ 525
***int*** vsobel(IMAGE *in, IMAGE *out, int weight_factor) ........ 526
***int*** warp_image(IMAGE *in, IMAGE *out) ........ 526
***int*** width(LIST *link) ........ 527
***int*** win_to_comp(COMPTR comptr, IMAGE *image, int left, int top, int sizex, int sizey) . 528
***int*** wrap(IMAGE *in, IMAGE *out, int hdispl, int vdispl, int zdispl) ........ 528
***unsigned int*** write(int fildes, char *buffer, int nbytes) ........ 529
***int*** write_var_object(VAR_OBJECT *object, char *filename) ........ 530
***int*** writefile(IMAGE *image, char *filename, int fileformat) ........ 531
***int*** xmax(LIST *link) ........ 532
***int*** xmin(LIST *link) ........ 533
***int*** xor_im(IMAGE *in1, IMAGE *in2, IMAGE *out) ........ 534
***void*** XYZ_ref_white(double Xn, double Yn, double Zn) ........ 535
***int*** ymax(LIST *link) ........ 536
***int*** ymin(LIST *link) ........ 537
***int*** z_planes(IMAGE *in, IMAGE *out, int start, int number, int border, int value) ........ 538
***int*** zcross(IMAGE *input, IMAGE *output, double threshold) ........ 538

# Index

## $

$ 522; 539

## :

: 231; 539

## ?

? 228; 539

## _

_exit 141

## A

abort 5
abs 6
abs_im 7
abs_pathname 37
absd_im 8
acos 467
acos_im 469
ActivateMenu 311
add_applic_exposure_func 9
add_applic_win_input_func 10
add_complex 11
add_float 12
add_im 13
add_int 14
AddImageInfo 15
AddToMenu 311
aim_readfile 16
aio_label 17
all_im 18
anchor_skelet 19
and_im 20
angle 20
angle_detection 21
applic_exposure 22
applic_win_input 23
apply_frequency_bank 24
apply_spatial_bank 24
arbit_dilation 25
arbit_erosion 26
area 26
asctime 27
asin 467
asin_im 469
atan 467
atan_im 469
atan2 467
atan2_im 469
atexit 29
atof 30
atoi 30
atol 30
auto_display 32
auto_plane 33
auto_point 34
average 35

## B

b_to_comp 35
back_project 36
bangle 36
bank_frequency_response 24
base_name 37
baskel 37
bcdist 38
bclose 38
bcont 38
bcount 38
bdila 39
bdist 39
bdskel 39
bedge 39
bend 40
benke 41
bernsen_threshold 100
beros 42
bi_threshold 496
bin_disp_colors 42
binary_to_grey 43
binary_to_plane 43
bit_ok 44
blabel 44
blife 44
bline 45
blow 45
bmaj 45
bopen 46
box_dimension 46
bperc 47
bprop 47
bpsr 47
bremh 47
bril 48

bsearch 48
bskbp 49
bskel 49
bskep 49
bsklp 49
bsngl 50
buf_from_silo 50
buf_to_silo 51

C

cabs 241
calc_greyvalue 52
calibrated_density 53
calloc 314
canny 54
cd 522; 539
cdens 55
ceil 146
chain 55; 539
chaincode_to_image 56
chaincode_to_xy 57
change_image_size 58
channel_bi_threshold 59
chdir 60
check_image_integrity 61
check_status 62
clear_im 63
clear_image_flag 431
clear_part_image 63
clear_var_object 64
clearerr 65
clip 66
clock 27
close 66
close_comp 67
close_silo 67
closing3x3 68
cmp_pixels 70
cnvo 70
color_get_model_size 71
color_set_color_model 71
com_dialog 72
compact_silo 73
complex_im 73
complx 74
compute_clut 74
con_ok 75
con6_ok 76
conjugate_im 76
conjugate_mul_im 77
constr_distance 78
contour 79
contrast_stretch 79
convert 80
convert_cmodel 81
convolution 82
cooccur 83
copy_channel 83
copy_clut 84
copy_histogram 92
copy_im 84
copy_masked_part 86
copy_object 85
copy_part_image 86
cos 467
cos_im 469
cosh 467
cosh_im 469
covariance 87
covmatrix 87
covplanematrix 88
cr 88
creat 89
create_clut 90
create_diff_window 95
create_display 91
create_histogram 92
create_image 94
create_live_window 95
create_silo 95
cst 96
ctime 27
cube_view 97

D

decrement_im 98
default_images 99
defuz 100
deiconify_window 393
del_im_exposure_func 249
del_im_input_func 249
dens 101
dens_limits 102
density 104
destroy_clut 105
destroy_histogram 92
destroy_image 105
destroy_var_object 106

dialog_options 107
dialog_stay_up 108
dialog_wm 108
different_ok 109
difftime 27
dilation3x3 110
dir_maximum 111
dir_name 37
disp_circle 112
disp_draw_mode 112
disp_draw_value 112
disp_oval 112
disp_rect 112
disp_srect 112
disp_text 112
disp_text_font 112
disp_vector 112
display_image 114
dist_average 115
dist_skelet 116
distance 117
dither 118
div_complex 11
div_float 12
div_im 13
div_int 14
do_alert 119
dots 120
draw_line 120
drawcurve 121
dump_histogram 235
dump_var_object 122
dyn_link 123
dyn_unlink 123

**E**

eccentr 124
ecvt 125
edge_average 126
edge_object 127
edge_ok 127
edge_preserve 128
edgps 128
eigen_segmentation 129
eigenfilters 130
eigenvectors 131
entropy_threshold 131
eql 132
equal_images 132
equalize 133
equivalent_im 133
eqv 134
erosion3x3 134
err_report 135
errno 360
eval 136
even_ok 139
EventType 140
exit 141
exp 142
exp_im 143
exp10_im 144
expand 145; 539

**F**

fabs 146
fast_fourier 147
fast_hartley 148
fblow 149
fclose 150
fcvt 125
fdopen 181
feof 65
ferror 65
fflush 150
fft 150
fg_buffers 151
fg_channels 152
fg_close 153
fg_depth 154
fg_exec 155
fg_freeze 161
fg_gain 156
fg_get_datasize 157
fg_get_last_error 158
fg_get_rgb_order 159
fg_getdata 160
fg_grab 161
fg_grab_image 162
fg_grab_next 163
fg_grab_series 164
fg_height 176
fg_init 165
fg_*load* 166
fg_maxdepth 154
fg_maxheight 167
fg_maxwidth 167
fg_minheight 168

fg_minwidth 168
fg_offset 169
fg_set_input_lut 170
fg_setdepth 171
fg_setgain 156
fg_setoffset 169
fg_setres 172
fg_setroi 173
fg_settype 174
fg_type 174
fg_width 176
fgetc 206
fgetpos 177
fgets 209
fgr_buffers 151
fgr_channels 152
fgr_close 153
fgr_depth 154
fgr_exec 155
fgr_freeze 161
fgr_gain 156
fgr_get_datasize 157
fgr_get_last_error 158
fgr_get_rgb_order 159
fgr_getdata 160
fgr_grab 161
fgr_grab_next 163
fgr_grab_series 164
fgr_height 176
fgr_init 165
fgr_maxdepth 154
fgr_maxheight 167
fgr_maxwidth 167
fgr_minheight 168
fgr_minwidth 168
fgr_offset 169
fgr_set_input_lut 170
fgr_setdepth 171
fgr_setgain 156
fgr_setoffset 169
fgr_setres 172
fgr_setroi 173
fgr_settype 174
fgr_type 174
fgr_width 176
fgreater0_ok 178
filter_energy_ratio 179
flip 179
floor 146
fmod 180
fopen 181
fpositive_ok 182
fprintf 383
fputc 390
fputs 391
fraction_im 183
frange_ok 184
fread 185
free 314
freopen 181
frexp 180
fscanf 418
fseek 186
fsetpos 177
fstat 482
ftell 186
funequal0_ok 187
fuz_width 188
fuzzy_derivative 188
fwrite 185

**G**

g_copy_object 190
g_copy_object_to 191
gabor_bank 309
gauss 192
gauss_deblur 193
gauss_family 194
gcvt 125
geo_affine 195
geo_rotate 195
geo_warp 195
get_bool_mask 197
get_display_mode 198
get_free_entry 198
get_im_error_stack 458
get_image_by_name 199
get_image_window_info 199
get_pixel 200
get_pixel_range 202
get_pixel_size 203
get_sizes 203
get_slice 204
get_super_clut 205
get_super_histo 205
get_super_im 205
getc 206
getchar 206

GETENV 207
GetImageInfo 15
getlogin 208
gets 209
getw 206
glc_asymmetry 210
glc_contrast 210
glc_entropy 210
gld_asymmetry 212
gld_contrast 212
gld_entropy 212
gld_mean 212
glr_greynonuniformity 214
glr_longrunemphasis 214
glr_nonuniformity 214
glr_percentage 214
glr_shortrunemphasis 214
gmtime 27
gravx 215
gravy 216
greater0_ok 217
greduce 218
grep 522; 539
grey_dilation 218
grey_erosion 219
grey_mean 220
grey_morph_arbit 221
grey_morph_diamond 221
grey_morph_ellipse 221
grey_morph_hollow_ellipse 221
grey_morph_round 221
grey_stdev 223
grey_sum 224

H

handle_events 432
handle_pim 225
have_diff 226
height 227
help 228; 539
hide_object 229
hide_object_at 229
hild_skelet 230
hist 231; 539
hist2d 232
histdata 233
histo_data 92
histogram 233
histogram_by_name 234
histogram_comment 235
histogram_ok 234
histogram_to_image 236
histogram_to_var_object 236
hit_or_miss 237
holt_skelet 238
homomorphic 239
hull 240
hypot 241

I

Ibenke 242
iconify_window 393
ics_readfile 243
ics_writefile 244
ifft 244
IGreyMap 245
im_begin_func 246
im_clear_errors 458
im_clear_func_stack 458
im_debug_stack 246
im_eigenvectors 277
im_end_func 246
im_exposure_func 249
im_from_silo 248
im_get_func_stack_copy 458
im_get_status 246
im_input_func 249
im_principle_component 277
im_report_error 246
im_set_output_handler 251
im_to_silo 252
im_val_ok 253
im1ps 254
im2ps 254
im3ps 254
im4ps 254
image_ok 255
image_output 256
image_readwrite_ok 257
image_text 257
image_to_chaincode 258
image_to_histogram 236
image_to_var_object 259
ImageMotionEvents 260
images_ok 261
imaginary_im 261
Imeasure 262
increment_im 263

index 486
init_func_overload 264
init_scil_image 264
init_silo 265
initimage 265
interpret 265
intlow 266
invert_im 266
Irectangle 267
is_histogram 234
is_image 268
is_var_object 269
isalnum 270
isalpha 270
isascii 270
iscntrl 270
isdigit 270
isgraph 270
islower 270
IsMouseDown 272
isodata_threshold 273
isprint 270
ispunct 270
isspace 270
isupper 270
isxdigit 270
iter_ok 274
IThreshold 275

## J

join_channels 478
jpeg_readfile 276
jpeg_writefile 276

## K

karhunen_loeve 277
KeyPressed 278
kirsch_temp 279
kuwahara 280
kuwahara_round 281

## L

label 282
labs 6
laplace 283
laxis 284
ldexp 180
lens 285
life 286
list 287; 539
list_cluts 288
list_histograms 235
list_label 289
list_var_objects 290
lmax 290
lmin 291
ln 291
load 292; 539
local_contrast 293
local_glc_asymmetry 295
local_glc_contrast 295
local_glc_entropy 295
local_gld_asymmetry 297
local_gld_contrast 297
local_gld_entropy 297
local_gld_mean 297
local_glr_greynonuniformity 299
local_glr_longrunemphasis 299
local_glr_nonuniformity 299
local_glr_percentage 299
local_glr_shortrunemphasis 299
localtime 27
log 142
log10 142
log10_im 300
logoff 301; 539
logon 301; 539
lookup 302
lower_gskeleton 303
lower_window 393
lowest_int 304
ls 522; 539
lseek 305

## M

macro 306; 539
majority 307
make_color_im 308
make_complex_im 73
make_gabor 309
make_image 310
MakeNewMenu 311
malloc 314
max_element 315
maximum_cost_path 316
maximum_im 317
maximum_trace 318
maxval 319
measure 320

memchr 321
memcmp 321
memcpy 321
memmove 321
memset 321
merge 322
merge_horizontal 322
merge_vertical 323
mergh 323
mergv 323
message_line_info 324
min_element 315; 316
minelm 325
minimum_im 325
minval 325
mirrh 326
mirror 326
mirror_horizontal 327
mirror_vertical 327
mirrv 328
mix 328
mix_filter 329
mktime 27
mmops3x3 330
modal_input 334
modf 180
modulo 335
more 287; 539
MouseMove 336
MousePress 337
MouseRelease 338
muj 339
mul_complex 11
mul_float 12
mul_im 13
mul_int 14

**N**

natural_window_size 339
nearest_int 340
negation_im 340
next_plane 341

**O**

object_contour 341
object_dens_meas 342
object_freeman_meas 344
object_moment_meas 345
object_rect_to_silo 346
object_shape_meas 347
objectsize 348
od_mean 220
od_stdev 223
od_sum 224
odd_fsizes_ok 349
odd_ok 350
open 351
open_silo 351
opening3x3 352
or_im 352
overload_func 353
overloadable_func 353

**P**

palette2color 354
parabolic_closing 355
parabolic_dilation 355
parabolic_erosion 355
parabolic_opening 355
part_from_silo 357
part_image_display 357
part_to_silo 358
perc_to_pixel 461
percentile 358
peri 359
perror 360
phase_im 361
pix_abs_sum 361
pix_average_val 362
pix_count 362
pix_maxval 363
pix_minval 363
pix_sum 364
pix_value_str 364
pixval 365
pl_io_ok 366
plane_ok 367
plane_to_binary 367
planecopy 368
plot_histogram 368
point_im 369
point_im_display_buf 370
point_object 371
poll_mouse 372
positive_ok 373
post_op 374
pow 142
power_im 374

power_of_2_ok 375
pre_op 376
prefered_color_model 71
prewd 379
prewitt_diff 380
prewitt_temp 381
prewt 382
print_RGB_matrices 436
print_XYZ_ref_white 535
printf 383
propagation 385
ps_head 386
ps_image 386
ps_tail 386
pseudo 388
psremoval 388
put 389
put_pixel 200
put_slice 204
put_xy_into_image 389
putc 390
putchar 390
puts 391
putw 390
pwd 522; 539

**Q**

qpix 392
qsort 48
quit 392

**R**

raise_window 393
rand 394
random_filter 394
random_im 395
range_ok 396
raster 397
read 398
read_var_object 399
readfile 400
real_im 401
real_time_recognizer 401
realloc 314
reduce 402
rel_searchfile 421
remark 403
remove 404
remove_holes 404
RemoveImageInfo 15
rename 404
resample_perp 405
retrieve_object_list 406
rewind 186
RGB_clear_extra 52
RGB_gamma_correction 406
RGB_ref_white 535
rhull 407
rindex 486
rm_list 407
rm_object 408
rm_silo 409
rm_silo_object 409
rmvar 410; 539
roberts_diff 410
robg 411
robinson_temp 412
roi_define 413
rotate 414
run 414; 539

**S**

S_Append 415
S_BreakList 415
S_CloseList 415
S_CopyList 415
S_Delete 415
S_FindItem 415
S_FreeList 415
S_Insert 415
S_Length 415
S_Prefix 415
S_SortList 415
saxis 417
scanf 418
searchfile 421
set 433
set_aio_disp 422
set_border 422
set_clut 423
set_color 423
set_color_model 71
set_common_line 424
set_comp_margin 425
set_complex 425
set_cross_dim 376
set_dialog_pos 426
set_display_mode 427

set_display_slice 428
set_dither_mode 429
set_float 430
set_im_type 430
set_image_flag 431
set_image_interaction 432
set_jpeg_quality 276
set_line_editor 434
set_menu_pos 435
set_rgb_bits 435
set_RGB2XYZ_matrix 436
set_RGB2XYZ_mvalues 436
set_roi_clean_display 437
set_roi_mask 438
set_roi_parent 438
set_roi_pos 438
set_screen_gamma 439
set_sigmoid_shape 440
set_start_pos 441
set_start_sizes 441
set_tiff_compression 442
set_tiff_image_number 442
set_var_object_class 443
set_var_object_comment 443
set_var_object_data 444
set_var_object_size 445
set_var_object_type 446
set_window_pos 447
set_window_size 447
setbuf 448
setbuffer 448
SETENV 207
setlinebuf 448
SetMacFileCreator 450
SetMacFileType 450
setprompt 451
setvbuf 448
sfp 452
sh 522; 539
shape 454
shift_im 455
show_cur_dir 456
show_dmode_flags 456
show_error_stack 458
show_func_overload 457
show_func_stack 458
show_histogram_info 235
show_image_info 459
show_menu_layout 460
show_statistics 461
show_var_object_info 463
sigma 463
sign_im 464
silo_list 465
silo_to_comp 466
sin 467
sin_im 469
single_pixels 470
sinh 467
sinh_im 469
size 471
skelpoints 471
sleep 472
small_object_removal 472
sobel_diff 473
sos 473
spb_publish 474
spb_subscribe 475
spb_unsubscribe 475
spix 476
splih 476
split 477
split_channels 478
split_color_im 308
split_horizontal 479
split_vertical 479
spliv 480
sprintf 383
sqrt 142
sqrt_im 480
srand 394
srandom_im 395
sscanf 418
ssum 480
standard_gabor 309
start_comp 481
stat 482
stereo_view 484
stop_bril 48
stop_lens 285
strcasecmp 485
strcat 486
strchr 486
strcmp 486
strcpy 486
strcspn 486
strerror 486
strftime 27

strlen 486
strncasecmp 485
strncat 486
strncmp 486
strncpy 486
strpbrk 486
strrchr 486
strspn 486
strstr 486
strtod 30
strtok 486
strtol 30
strtoul 30
sub_complex 11
sub_float 12
sub_im 13
sub_int 14
swab 488
sync_display 489
system 489

**T**

t_morphology 490
tan 467
tan_im 469
tanh 467
tanh_im 469
taylor_expansion 492
taylor_polynomial 492
taylor_segmentation 493
tcl_readfile 494
tcl_writefile 495
tell 305
threshold 496
ti_block 497
ti_circle 497
ti_fractal 498
ti_hlines 499
ti_ifr 498
ti_lines 500
ti_lshading 500
ti_points 501
ti_qshading 501
ti_vlines 502
ticb 502
ticc 502
tiff_readfile 503
tiff_writefile 504
tilh 504
tiln 505
tils 505
tilv 505
time 27; 506; 539
tipt 506
tiqs 506
tmpfile 507
tmpnam 507
tolower 508
toupper 508
trans_mean 220
trans_stdev 223
trans_sum 224
tri_state_threshold 509
truncate_im 510
TWAcquire 511
TWAcquireArea 511
txt 512
type_overload_func 353
type_overloadable_func 353

**U**

unequal0_ok 512
ungetc 513
uniform 514
uniform_round 514
update 515
upper_gskeleton 303

**V**

val_check 516
var_object 517
var_object_by_name 518
var_object_convert 519
var_object_copy 519
var_object_ok 520
var_object_to_image 521
vi 522; 539
vkuwahara 523
vlaplace 524
vlinear 524
vmedian 525
vpercentile 525
vsobel 526

**W**

warp_image 526
width 527
win_to_comp 528
wrap 528

write 529
write_var_object 530
writefile 531
writf 531

X

xmax 532
xmin 533
xor_im 534
XYZ_ref_white 535

Y

ymax 536
ymin 537

Z

z_planes 538
zcross 538