The
Connection Machine
System

Image File Interface Reference Manual for Paris

Version 2.0
November 1991

Thinking Machines Corporation
Cambridge, Massachusetts

First printing, November 1991

****

The information in this document is subject to change without notice and should not be construed as a commitment by Thinking Machines Corporation. Thinking Machines Corporation reserves the right to make changes to any products described herein to improve functioning or design. Although the information in this document has been reviewed and is believed to be reliable, Thinking Machines Corporation does not assume responsibility or liability for any errors that may appear in this document. Thinking Machines Corporation does not assume any liability arising from the application or use of any information or product described herein.

****

Thinking Machines Corporation
245 First Street
Cambridge, Massachusetts 02142–1264
(617) 234–1000/876–1111

Contents

About This Manual

Objectives of This Manual

This manual provides user and reference information for the Paris interface to the Version 2.0 release of the Image File Interface. Separate manuals are available for the C* and CM Fortran interfaces.

Intended Audience

This manual is intended for programmers using the Image File Interface. The reader is assumed to be familiar with basic Paris programming.

Revision Information

This is the first release of this manual.

Organization of This Manual

Chapter 1 **The Image File Interface**
Provides a brief introduction to Image File Interface, including an overview of the routines and information on how to use them.

Chapter 2 **The Image File Interface Routines**
Provides separate detailed descriptions of each routine.

Related Documents

This manual is one of three that make up the Connection Machine Visualization Programming documentation set. The other two are:

- *Generic Display Interface Reference Manual for Paris*
- **Render Reference Manual for Paris*

Notation Conventions

The table below displays the notation conventions observed in this manual.

Convention	Meaning
`bold typewriter`	C/Paris and Fortran/Paris language elements, such as operators, keywords, and function names, when they appear embedded in text or in syntax lines. Also UNIX and CM System Software commands, command options, and file names.
italics	Argument or parameter names and placeholders, when they appear embedded in text or syntax lines.
ANY_FE_ARRAY	Signals that the array may be any data type on the front end.
`typewriter`	Code examples and code fragments.

`%` **`bold typewriter`**
`typewriter`
	In interactive examples, user input is shown in **`bold typewriter`** and system output is shown in `regular typewriter` font.

Customer Support

Thinking Machines Customer Support encourages customers to report errors in Connection Machine operation and to suggest improvements in our products.

When reporting an error, please provide as much information as possible to help us identify and correct the problem. A code example that failed to execute, a session transcript, the record of a backtrace, or other such information can greatly reduce the time it takes Thinking Machines to respond to the report.

If your site has an Applications Engineer or a local site coordinator, please contact that person directly for support. Otherwise, please contact Thinking Machines' home office customer support staff:

U.S. Mail:
Thinking Machines Corporation
Customer Support
245 First Street
Cambridge, Massachusetts 02142–1264

Internet Electronic Mail: customer–support@think.com

uucp Electronic Mail: ames!think!customer–support

Telephone:
(617) 234–4000
(617) 876–1111

For Symbolics Users Only

The Symbolics Lisp machine, when connected to the Internet network, provides a special mail facility for automatic reporting of Connection Machine system errors. When such an error occurs, simply press Ctrl–M to create a report. In the mail window that appears, the `To:` field should be addressed as follows:

`To:` customer–support@think.com

Please supplement the automatic report with any further pertinent information.

Chapter 1

The Image File Interface

1.1 Overview

The Image File Interface allows you to store image data from the CM system in files for later display or processing. This interface includes routines to transfer image data to and from an image data file and

- an image buffer in CM memory
- an array on the front-end computer
- a generic display (i.e., a CM framebuffer or an X11 window initialized as a Generic Display Interface display)

The Image File Interface writes and reads files that conform to the the TIFF (Tagged Image File Format) 5.0 specification as published by Aldus Corporation and Microsoft Corporation. This format is accepted by many other graphics systems and software packages on platforms ranging from personal computers to supercomputers. Thus, this interface makes it possible for you to move images between the CM system and many other graphics environments.

The TIFF format allows you to store image data in a compressed format and also to store information about the image and its display environment in the file. When the image is later read back into the CM system by the Image File Interface, or read by another TIFF reader elsewhere, this information allows the software to interpret and display the data correctly.

1.1.1 Including Image File Interface Routines in a Program

To use the Image File Interface routines, you must include the appropriate header file in your program and link with the supporting libraries when compiling.

For C/Paris programs you must

- include the the header file **cmtiff.h**:

```
#include <cm/cmtiff.h>
```

- use the following links:

```
cc prog.c -lcmsr -ltiff -lX11 -lparis -lm
```

For Fortran/Paris programs you must

- include the the header file **cmtiff-fort.h**:

```
INCLUDE '/usr/include/cm/cmtiff-fort.h>
```

- use the following links:

```
f77 prog.f -lcmsr -ltiff -lX11 -lparisfort -lparis -lm
```

For Lisp programs you must use a band in which the graphics package has been loaded. If necessary, you can load it by entering

```
(lcmw:load-optional-system 'graphics)
```

1.2 Image Transfer

The Image File Interface provides a simple functional interface to read or write image data between a TIFF-formatted file and an image buffer, a front-end array, or a generic display.

1.2.1 Writing an Image to a File

For example, to store an image from a generic display in a TIFF-formatted file, you can use a single CMSR (CM *Render) subroutine call:

CMSR_image_display_to_file_simple
(filename, append_p, width, height)

This routine writes an image array from the currently selected generic display to the file named. The area of the display transferred is an array of *width* by *height* pixels, beginning from the point defined by any Generic Display Interface offsets in effect for the display, or from the origin of the display space if no offsets are set. If *append_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp), the image is added to the end of the file after any other images already stored there; if *append_p* is false (.FALSE. in Fortran, NULL in C, nil in Lisp), this image overwrites any that may be in the file. **`CMSR_image_display_to_file_simple`** also reads information about the image, such as pixel depth and color type, from the Generic Display and stores it in the file with the image data. A default compression strategy is used to store the image data.

Two similar routines write images to a file from an image field in CM memory and from an image array in the front-end computer's memory:

- **`CMSR_image_field_to_file_simple`**
  (**filename, src_field, slen, append_p, x_varies_fastest_p, width, height, photometric_interpretation, samples_per_pixel, bits_per_sample, red*[ ], *green*[ ], *blue*[ ])

- **`CMSR_image_array_to_file_simple`**
  (**filename, *image_array, append_p, width, height, photometric_interpretation, samples_per_pixel, bits_per_sample, red*[ ], *green*[ ], *blue*[ ])

Because these routines cannot determine the image characteristics as **`CMSR_image_display_to_file_simple`** can from the generic display, you must specify the essential image information as arguments to the routine.

To avoid specifying these parameters each time you save an image, you can use one of a set of routines that use an *image information structure* to define the image characteristics and also allow you to control the compression and configuration strategies used to store the image data in the file:

- **`CMSR_image_array_to_file`**
  (**filename, *image_array, image_info, append_p*)

- **`CMSR_image_display_to_file`** (**filename, image_info, append_p*)

- **`CMSR_image_field_to_file`**
  (**filename, src_field, slen, image_info, append_p, x_varies_fastest_p*)

The image information structure is an Image File Interface data structure used to maintain information about the type of image to be stored and way the data is to be organized in the file. In addition, you can record the image artist, date and time of creation, the host computer, and the software used. The Image File Interface provides routines that allocate this structure, and that set and read back its parameters. This allows you to control the way the image is stored, for example, to match the characteristics of another TIFF reader. See Section 1.3 below for a more detailed discussion.

1.2.2 Reading an Image from a File

A similar set of routines transfers image data from a TIFF-formatted file to a CM field, an array on the front-end computer, or directly to a generic display:

- **`CMSR_image_file_to_field`**
  (**filename, dest_field, dlen, image_info, image_number, x_varies_fastest_p*)
- **`CMSR_image_file_to_array`**
  (**filename, image_array, image_info, image_number*)
- **`CMSR_image_file_to_display`** (**filename, image_info, image_number*)

`CMSR_image_file_to_field` and **`CMSR_image_file_to_array`** read the image by transferring each pixel's data from the file to the corresponding element of the 2D field or array. **`CMSR_image_file_to_display`** transfers the image data to the currently selected generic display beginning at the origin (0,0) of the display space or at the point specified by any Generic Display Interface offsets that may be set.

The *image_info* argument specifies an *image information structure*. When these routines read in the image, they also load the *image_ info* structure with the information about the image stored in the TIFF file. The Image File Interface then uses this information to interpret the image data properly. You can also access the information to configure your display environment. If you leave the *image_info* argument as NULL or zero, the *image_info* argument is ignored and no data describing the image is returned.

If more than one image is stored in the file, the *image_number* argument specifies which image is to be returned.

1.2.3 Image Transfer Commands

In addition, two shell-level commands allow you transfer an image between a file and a CM framebuffer generic display:

- **cmdisplay2tiff** [-append] [-x *offset*] [-y *offset*] [-w *width*]
  [-h *height*] [-artist *name*] [-host *name*]
  [-description *string*] [-software *name*]
  [-separate] [-nocomp | -lzw]
  [-rowsperstrip *number*] *filename*

- **tiff2cmdisplay** [-x *offset*] [-y *offset*] [-image *number*]
  [-v] [-nozoom] [-nowait] *filename*

1.3 The Image Information Structure

The Image File Interface routines use the image information structure to interpret data when transferring images to or from an image file. When an image is written from the CM system to a file, the transfer routines use this information to organize the image data in the file and then store it with the image data. When a stored image is read back into the CM system, these routines automatically load this information into an image information structure and you can use it to configure the display or array to which the image is sent.

If you usually use a standard image definition and file format or have a series of similarly defined images to store, the image information structure allows you to define a standard set of specifications that can be referenced by the Image File Interface routines. If you have only a small number of images of a certain type, the image transfer routines ending in **-simple** do not require an image information structure. These routines allow you to specify a minimum number of defining parameters as arguments.

The Image File Interface provides routines to allocate and deallocate this structure:

- **CMSR_allocate_image_info** ()

- **CMSR_deallocate_image_info** (*image_info*)

A routine is also provided that reads the image information from a file into an image information structure:

- **CMSR_image_get_info** (*filename, image_info, image_number*)

1.3.1 Image Attributes Information

An image is described in the image information structure by

- image width and height in pixels
- number of color samples per pixel
- number of bits per sample
- photometric interpretation: how the image samples are to be interpreted
- color map arrays to be installed with the image.

You use Image File Interface routines to set these parameters and to return the current setting.

Image Width and Height

The image width is set by **`CMSR_image_set_width`** (*image_info*, *width*) and the current setting of image width is returned by **`CMSR_image_width`** (*image_info*). Similarly, the image height is set by **`CMSR_image_set_height`** (*image_info*, *height*) and returned by **`CMSR_image_height`** (*image_info*). The width of the image is the number of pixels in the horizontal (*x*) dimension and the height is the vertical (*y*) dimension in pixels.

Color Samples

The number of color samples per pixel is set by **`CMSR_image_set_num_sample`** (*image_info*, *samples_per_pixel*) and returned by **`CMSR_image_num_samples`** (*image_ info*). The number of samples is the number of color components maintained for each pixel. For example, monochrome, grayscale, and pseudo-color images all maintain one sample per pixel, while true color (direct color) images maintain three samples, usually red, green, and blue.

The number of bits per sample is set by **`CMSR_image_set_bits_per_sample`** (*image_info*, *bits_per_sample*) and returned by **`CMSR_image_bits_per_sample`** (*image_info*). The number of bits per sample specifies the size of each component. For example, a monochrome image is defined in 1 bit per sample; grayscale or pseudo color images are often 8 bits per sample: RGB images are usually composed of three 8-bit samples. (For images composed of more than one sample per pixel, the Image File Interface requires that the number of bits be the same for all the samples.)

Photometric Interpretation

The photometric interpretation indicates how the pixel data is to be interpreted. For example, an image with one sample per pixel and 8 bits per sample may be interpreted as either a grayscale or a palette (pseudo color) image. This parameter is set by **`CMSR_image_set_photometric`** *(image_info, photometric)* and returned by **`CMSR_image_photometric`** *(image_info)*.

The TIFF photometric interpretations supported by the Image File Interface are

- **`PHOTOMETRIC_MINISWHITE`** and **`PHOTOMETRIC_MINISBLACK`** for monochrome and grayscale images. These values specify whether the minimum value in the color map is white or black.
- **`PHOTOMETRIC_PALETTE`** indicates the image will contain one sample per pixel, which is interpreted as an index into a color map.
- **`PHOTOMETRIC_RGB`** indicates three or four samples per pixel. The first three samples are interpreted as the red, green, and blue intensities of the color. You can use the fourth sample for whatever purpose you wish. For example, the fourth sample may be used to store *z*-buffer values or a transparency mask.

Color Map

You must specify a color map for images with a **`PHOTOMETRIC_PALETTE`** photometric interpretation. The color map is set with **`CMSR_image_set_color_map`** *(image_info, red, green, blue)* and the color map currently set in an image information structure is returned with **`CMSR_image_color_map`** *(image_info)*.

When you transfer a palette color image to a generic display, the display is automatically set to the color map stored with the image in the *red*, *green*, and *blue* arrays.

1.3.2 Image Environment Notes

In addition to the image attributes, you can store information about the context in which the image was created as a part of the TIFF file itself.

The image information structure accepts character strings in which you can record

- the image artist
- date and time the image was created

- host computer (computer on which the image was created)
- software used to create the image
- text description of the image

As with the attributes, there are routines to set and read back each of these fields:

- **CMSR_image_set_artist** (*image_info, *string*)
  CMSR_image_artist (*image_info*)
- **CMSR_image_set_date_time** (*image_info, *string*)
  CMSR_image_date_time (*image_info*)
- **CMSR_image_set_host_computer** (*image_info, *string*)
  CMSR_image_host_computer (*image_info*)
- **CMSR_image_set_software** (*image_info, *string*)
  CMSR_image_software (*image_info*)
- **CMSR_image_set_description** (*image_info, *string*)
  CMSR_image_description (*image_info*)

1.3.3 File Format Information

Information about the file format that is stored for TIFF files includes

- compression strategy
- planar configuration (how the color planes are to be organized in the file)
- rows per strip (the number of rows [scanlines] of image data to be stored for each strip of data in the file)

Compression Strategy

The compression parameter allows you choose to store the image data in the file in a compressed format or not.

The compression parameter lets you select different compression strategies depending on the device that will be used to read the file. The supported compression strategies are

- **COMPRESSION_LZW**
- **COMPRESSION_NONE**

The default is the general compression strategy **LZW**. **LZW** uses the Lempel–Ziv and Welch algorithm for data compression. This is a general purpose algorithm that works well on any type of image. The algorithm is described in detail in Appendix F of the TIFF (Tagged Image File Format) 5.0 specification as published by Aldus Corporation and Microsoft Corporation. (See Section 1.5)

Most other TIFF readers accept the **LZW** compression strategy. However, if you are transferring the image to another graphics environment, check on the compression strategies supported by the software you will be using there. If you do not know what compression methods are supported, we recommend no compression (**NONE**). Images stored with no compression should be acceptable to nearly all TIFF readers.

Planar Configuration

The configuration parameter describes how image data that contains more than one sample per pixel is to be stored. The configuration strategy is set with **CMSR_image_set_planar_config** (*image_info*, *configuration)* and returned with **CMSR_image_planar_config** (*image_info*)**.**

The image samples can be stored either as a contiguous array (**PLANARCONFIG_CONTIG**) or as one array for each separate plane of samples (**PLANARCONFIG_SEPARATE**). For example, an RGB image stored contiguously would have the samples interleaved: RGBRGBRGB.... The same image stored separately would have one plane for the red samples, another for the green samples, and a third for the blue samples.

Your choice of configuration methods will depend on whether you will be porting the image to other software and what compression strategy you wish to use. Some software packages will only accept contiguous images because each pixel's values are available in sequence. However, many compression strategies work more efficiently when the color samples are stored in separate planes.

The configuration parameter is not used for images with only one sample per pixel.

Rows per Strip

The rows per strip parameter describes the number of rows (scanlines) of image data that is to be stored for each strip of data in the image file. The number is set with **`CMSR_image_set_rows_per_strip`** `(`*image_info, rows_per_strip*`)` and returned with **`CMSR_image_rows_per_strip`**`(`*image_info*`)`.

This parameter allows you to control the amount of data that will be sent with each read from the file. This should be adjusted to allow efficient buffering on your system. If not specified, the Image File Interface sets the parameter to result in about 8K of data being buffered at a time.

1.4 Supported TIFF Classes

The following lists the TIFF image classes supported by the Image File Interface.

TIFF Class B (Bilevel/Monochrome)

- Samples per pixel: 1
- Bits per sample: 1
- Photometric Interpretation: **`PHOTOMETRIC_MINISWHITE`** or **`PHOTOMETRIC_MINISBLACK`**
- Compression: **`COMPRESSION_NONE`** or **`COMPRESSION_LZW`**

This photometric indicates whether the low entry in the color map is black or white. For **`PHOTOMETRIC_MINISWHITE,`** for example, pixels with a value of 0 are displayed as white and pixels with a value of 1 are black.

TIFF Class G (Grayscale)

- Samples per pixel: 1
- Bits per sample: 4 or 8
- Photometric Interpretation: **`PHOTOMETRIC_MINISWHITE`** or **`PHOTOMETRIC_MINISBLACK`**

- Compression: **COMPRESSION_NONE** or **COMPRESSION_LZW**

This photometric indicates whether the low entry in the color map is black or white. For **PHOTOMETRIC_MINISWHITE,** for example, the color ramp is a range of gray intensities running from white at 0 to black at the maximum color map entry.

TIFF Class P (Palette, Pseudo Color)

- Samples per pixel: 1
- Bits per sample: 1, 2, 4, 8
- Photometric Interpretation: **PHOTOMETRIC_PALETTE**
- Compression: **COMPRESSION_NONE** or **COMPRESSION_LZW**

When photometric interpretation is set to indicate a palette color image, you must also supply a color map.

NOTE: The TIFF 5.0 specification allows for any number of bits per sample in the palette class from 1 to 8. The Image File Interface supports only 1, 2, 4, or 8 bits, as listed above.

TIFF Class R (RGB Full Color)

- Samples per pixel: 3 or 4
- Bits per sample: 8
- Photometric Interpretation: **PHOTOMETRIC_RGB**
- Compression: **COMPRESSION_NONE** or **COMPRESSION_LZW**

NOTE: TIFF 5.0 specifies only 3 samples for the Class R images. The Image File Interface extends this to allow four samples. When an image with a photometric interpretation of **PHOTOMETRIC_RGB** and four samples is transferred to a CM field or front-end array, the fourth sample is simply written to memory following the other three samples. You must have allocated sufficient memory for the additional sample. If the image is transferred to a generic display, the fourth sample is not used. You may use the fourth sample in your application in any way you wish, for example, as a *z*-buffer or a transparency mask.

1.5 TIFF 5.0 Information

For detailed information on the TIFF 5.0 specification, contact either of the following:

- Developer's Desk, Aldus Corporation
  411 First Ave. So., Suite 200 Box 97017
  Seattle, WA 98104 Tel: (206) 622–5500

- Windows Marketing Group, Microsoft Corporation
  16011 NE 36th Way
  Redmond, WA 98073–9717 Tel: (206) 882–8080

Much of the Image File Interface is built on top of a library of support routines that manipulate TIFF files. This library, along with several useful tools and test images, is available by anonymous **ftp**.

To retrieve a copy, you can use **`ftp`** to connect to either **`ucbvax.berkeley.edu (128.132.130.12)`** or **`uunet.uu.net (192.48.96.2)`**. From **ucbvax**, you should retrieve the file **`pub/tiff/v2.4.tar.Z`**, from **`uunet`** retrieve **`graphics/tiff.tar.Z`** . Please remember that these machines are heavily used, and try to retrieve files from them outside of normal business hours.

The file that you get will be a compressed tar file, so remember to set binary mode when using **`ftp`**. For example:

```
% ftp ucbvax.berkeley.edu
Connected to ucbvax.berkeley.edu.
220 ucbvax.Berkeley.EDU FTP server
Name (ucbvax.berkeley.edu:matt): anonymous
331 Guest login ok, send ident as password.
Password:
230 Guest login ok, access restrictions apply.
ftp> binary
200 Type set to I.
ftp> get v2.4.tar.Z
200 PORT command successful.
150 Opening BINARY mode data connection for v2.4.tar.Z (333827 bytes).
226 Transfer complete.
local: v2.4.tar.Z remote: v2.4.tar.Z
333827 bytes received in 25 seconds (13 Kbytes/s)
ftp> quit
221 Goodbye.
```

The software comes in a compressed tar file. To extract the information:

Chapter 2

Image File Operations

This chapter provides detailed descriptions of the Image File Interface routines divided into two groups:

- Image File Transfer routines (beginning immediately below)
- Image Information Structure routines (beginning on page 44)

2.1 Image Transfer Routines

This section describes the Image File Transfer routines that read and write files between the CM system and an image file.

The Image File Transfer routines read and write files between the CM system and an image file. These routines are:

CMSR_image_array_to_file

Transfers an image from a front-end array to an image file.

SYNTAX

C Syntax

```
#include <cm/cmtiff.h>

int
  CMSR_image_array_to_file
                         (*filename, *image_array, image_info, append_p)

char                     *filename;
CMSR_generic_pointer_t   image_array;
CMSR_image_info_t        image_info;
int                      append_p;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmtiff-fort.h'

INTEGER FUNCTION CMSR_IMAGE_ARRAY_TO_FILE
&                        (filename, image_array, image_info, append_p)

   CHARACTER*(*)  filename
   ANY_FE_ARRAY   image_array
   INTEGER        image_info
   LOGICAL        append_p
```

ARGUMENTS

filename The name of the file to which the image is to be written.

image_array The front-end array from which the image is to be read. The image array may be of any data type. The array elements will be interpreted as image pixels according to the values provided in the fields for *width*, *height*, *samples_per_pixel*, *bits_per_sample*, and *photometric_interpretation* in the image information structure.

In C the **`CMSR_generic_pointer_t`** data type accepts a pointer to any data type. It is used to make code portable between ANSI compilers that accept pointers to **`void`** and VAX compilers that would require a pointer to **`char`** for unspecified data types. You may need to cast this variable to the data type you use for your array to avoid compiler warnings.

image_info

A **`CMSR_image_info_t`** data structure containing specifications for the image and for the format in which it is to be stored.

Image information structures are created with **`CMSR_allocate_image_info`**.

NOTE: There are no defaults in the image information structure for *width*, *height*, *photometric_interpretation*, *num_samples*, or *bits_per_sample*. Before you write an image you must make sure that these fields are set correctly by calling the appropriate Image File Interface Routine:

- **`CMSR_image_set_width`**
- **`CMSR_image_set_height`**
- **`CMSR_image_set_num_samples`**
- **`CMSR_image_set_bits_per_sample`**
- **`CMSR_image_set_photometric`**

append_p

A predicate indicating whether the image is appended to *filename* or overwrites *filename*. If *append_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp) the image from the array is appended to the end of the file. If *append_p* is false (.FALSE. in Fortran, NULL in C, nil in Lisp), the image from the field overwrites any images already in the file.

DESCRIPTION

`CMSR_image_array_to_file` transfers an array on the front-end computer to an image file.

The image array may be of any data type. But, before writing the contents of the array to a TIFF file, the *image_info* structure must be set to describe that image. At a minimum, this means that the width, height, samples per pixel, bits per sample, and photometric interpretation must be specified.

`CMSR_image_array_to_file` returns non-negative integers on success and `CMSR_IMAGE_ERROR` if an error is detected.

ERRORS

An error is signalled if

the specified file can not be opened for writing

an error occurs while writing scanlines to the file

SEE ALSO

`CMSR_image_array_to_file_simple`

CMSR_image_array_to_file_simple

Transfers a front-end array to an image file.

SYNTAX

C Syntax

```
#include <cm/cmtiff.h>

int
  CMSR_image_array_to_file_simple
         (*filename, *image_array, append_p, width, height,
          photometric_interpretation,
          samples_per_pixel, bits_per_sample, red, green, blue)

char                      *filename;
CMSR_generic_pointer_t    image_array;
int                       append_p;
int                       width, height;
int                       photometric_interpretation;
int                       samples_per_pixel, bits_per_sample;
float                     red[], green[], blue[];
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmtiff-fort.h'

   INTEGER FUNCTION CMSR_IMAGE_ARRAY_TO_FILE_SIMPLE
&               (*filename, *image_array, append_p, width, height,
&                photometric_interpretation,
&                samples_per_pixel, bits_per_sample, red, green, blue)

   CHARACTER*(*)  filename
   ANY_FE_ARRAY   image_array
   LOGICAL        append_p
   INTEGER        width, height
   INTEGER        photometric_interpretation
   INTEGER        samples_per_pixel bits_per_sample
   REAL           red(), green(), blue()
```

ARGUMENTS

filename The name of the file to which the image is to be written.

image_array The front-end array from which the image is to be read. The image array may be of any data type. The array elements will be interpreted as image pixels according to the values provided for *width*, *height*, *samples_per_pixel*, *bits_per_sample*, and *photometric_interpretation*.

In C the **`CMSR_generic_pointer_t`** data type accepts a pointer to any data type. It is used to make code portable between ANSI compilers that accept pointers to **`void`** and VAX compilers that would require a pointer to **`char`** for unspecified data types. You may need to cast this variable to the data type you use for your array to avoid compiler warnings.

append_p A predicate indicating whether the image is appended to *filename* or overwrites *filename*. If *append_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp), the image from the array is appended to the end of the file. If *append_p* is false (.FALSE. in Fortran, NULL in C, nil in Lisp), the image from the field overwrites any images already in the file.

width An integer specifying the number of pixels in the image's *x* (horizontal) dimension.

height An integer specifying the number of pixels in the image's *y* (vertical) dimension.

photometric_interpretation

An enumerated variable specifying how the image data is to be interpreted by Image File Interface.

Possible values can be:

- **`PHOTOMETRIC_MINISWHITE`**
- **`PHOTOMETRIC_MINISBLACK`**
- **`PHOTOMETRIC_RGB`**
- **`PHOTOMETRIC_PALETTE`**

samples_per_pixel The number of color components that are maintained per pixel in the image to be stored. Currently, the number of samples supported by the Image File Interface is the following:

- 1 for bilevel, grayscale, or palette images or image masks
- 3 (red, green, and blue) for RGB true color images
- 4 (red, green, blue, and alpha) for RGB plus alpha channel images

bits_per_sample The number of bits of color information maintained for each sample per pixel in the image. For one sample per pixel the bits per sample may be 1, 2, 4, or 8.

For three or four samples per pixel, the bits per sample must be 8. For the most common classes of images, the following numbers of bits per sample are supported:

- Bilevel images must be 1 bit per sample
- Grayscale images can be 4 or 8 bits per sample
- Palette color images can be 1, 2, 4, or 8 bits per sample
- RGB full-color images must be 8 bits per sample
- RGB plus alpha images must be 8 bits per sample

red, green, blue Arrays of color values specifying the red, green, and blue components of a color map. The arrays must each have 2^(*samples_per_pixel*) elements.

If the photometric interpretation is **`PHOTOMETRIC_PALETTE`**, the color map must be specified. For palette images, the color map is stored in the file with the image.

NOTE: If you want to load a color map from a generic display, you can use the Generic Display Interface routine **`CMSR_display_read_color_map`**.

DESCRIPTION

`CMSR_image_array_to_file_simple` transfers an image array on the front-end computer to an image file.

This routines returns non-negative integers on success and **`CMSR_IMAGE_ERROR`** if an error is detected.

ERRORS

An error is signalled if

the specified file cannot be opened for writing

an error occurs while writing scanlines to the file

SEE ALSO

`CMSR_image_array_to_file`

CMSR_image_display_to_file

Transfers an image from current CM Generic Display to a specified image file.

SYNTAX

C Syntax

```
#include <cm/cmtiff.h>

int
  CMSR_image_display_to_file
                   (*filename, image_info, append_p)

char               *filename;
CMSR_image_info_t  image_info;
int                append_p;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmtiff-fort.h'

   INTEGER FUNCTION CMSR_IMAGE_DISPLAY_TO_FILE
&                   (filename, image_info, append_p)

   CHARACTER*(*)    filename
   INTEGER          image_info
   LOGICAL          append_p
```

ARGUMENTS

filename The name of the file to which the image is to be written.

image_info A `CMSR_image_info_t` data structure containing specifications for the image and for the format in which it is to be stored.

append_p A predicate indicating whether the image is appended to *filename* or overwrites *filename*. If *append_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp), the image from the array is appended to the end of the file. If *append_p* is false (.FALSE. in Fortran, NULL in C, nil in Lisp), the image from the field overwrites any images already in the file.

DESCRIPTION

`CMSR_image_display_to_file` transfers an image directly from the current Generic Display display to an image file.

If the *image_info* structure does not contain entries for the image's width, height, bits per sample, samples per pixel, or photometric interpretation, these are determined from the currently selected generic display.

If the photometric interpretation is `PHOTOMETRIC_PALETTE`, the color map of the Generic Display display is stored with the image.

`CMSR_image_display_to_file` returns non-negative integers on success and `CMSR_IMAGE_ERROR` if an error is detected.

SEE ALSO

`CMSR_image_disp_to_file_simple`

CMSR_image_disp_to_file_simple

Transfers an image from current CM Generic Display to a specified image file without using image information structure.

SYNTAX

C Syntax

```
#include <cm/cmtiff.h>

int
  CMSR_image_disp_to_file_simple
                              (*filename, append_p, width, height)

char  *filename;
int   append_p;
int   width, height;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmtiff-fort.h'

   INTEGER FUNCTION CMSR_IMAGE_DISP_TO_FILE_SIMPLE
&                                (filename, append_p, width, height)

   CHARACTER*(*)  filename
   LOGICAL        append_p
   INTEGER        width, height
```

ARGUMENTS

filename

The name of the file to which the image is to be written.

The *filename* parameter must point to an area of memory large enough to hold the image.

append_p

A predicate indicating whether the image is appended to *filename* or overwrites *filename*. If *append_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp), the image from the array is appended to the end of the file. If *append_p* is false (.FALSE. in

Fortran, NULL in C, nil in Lisp), the image from the field overwrites any images already in the file.

width An integer specifying the number of pixels in the image's *x* (horizontal) dimension.

height An integer specifying the number of pixels in the image's *y* (vertical) dimension.

DESCRIPTION

`CMSR_image_disp_to_file_simple` transfers an image directly from the current Generic Display display to an image file.

The photometric interpretation is determined from the current Generic Display.

This routine returns a non-negative integer on success and `CMSR_IMAGE_ERROR` if an error is detected.

SEE ALSO

`CMSR_image_display_to_file`

`cmdisplay2tiff`

CMSR_image_field_to_file

Transfers an image from CM memory to a file using an image information structure.

SYNTAX

C Syntax

```
#include <cm/cmtiff.h>

int
   CMSR_image_field_to_file
        (*filename, src_field, src_len, image_info, append_p,
         x_varies_fastest_p)

char                *filename;
CM_field_id_t       src_field;
unsigned int        src_len;
CMSR_image_info_t   image_info;
int                 append_p;
int                 x_varies_fastest_p;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmtiff-fort.h'

   INTEGER FUNCTION CMSR_IMAGE_FIELD_TO_FILE
&      (filename, src_field, src_len, image_info, append_p, x_varies_fastest_p)

   CHARACTER*(*)     filename
   INTEGER           src_field
   INTEGER           src_len
   INTEGER           image_info
   LOGICAL           append_p
   LOGICAL           x_varies_fastest_p
```

ARGUMENTS

filename The file to which the image is to be written.

src_field The field ID of the field in CM memory from which the image is to be read.

src_len The length, in bits, of the source field containing the image.

image_info A **`CMSR_image_info_t`** data structure containing specifications for the image and for the format in which it is to be stored.

Image information structures are created with **`CMSR_allocate_image_info`**.

NOTE: There are no defaults in the image information structure for *width*, *height*, *photometric_interpretation*, *num_samples*, or *bits_per_sample*. Before you write an image you must make sure that these fields are set correctly by calling the appropriate Image File Interface routine:

- **`CMSR_image_set_width`**
- **`CMSR_image_set_height`**
- **`CMSR_image_set_num_samples`**
- **`CMSR_image_set_bits_per_sample`**
- **`CMSR_image_set_photometric`**

append_p A predicate indicating whether the image is appended to *filename* or overwrites *filename*. If *append_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp), the image from the array is appended to the end of the file. If *append_p* is false (.FALSE. in Fortran, NULL in C, nil in Lisp), the image from the field overwrites any images already in the file.

x_varies_fastest_p A boolean or logical specifying whether the image is stored such that the *x* or *y* coordinate varies fastest. Normally, *x_varies_fastest_p* should be set to true.

If *x_varies_fastest_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp), the image data is organized so that the *x* coordinate varies fastest, that is, in column-major order. This is the case for Fortran language arrays, and for C arrays that are referenced [*y*][*x*].

If *x_varies_fastest_p* is false (.FALSE. in Fortran, NULL in C, nil in Lisp), the image data is organized so that the *y* coordinate varies fastest, that is, in row-major order. This is the case for C arrays referenced [*x*][*y*].

The *Render display routines assume that *x* varies fastest and maps axis 0 of the image field to the *x* dimension of the display.

If your image arrays are stored so that *y* varies fastest, *x_varies_fastest* should be set to FALSE.

DESCRIPTION

CMSR_image_field_to_file transfers an image directly from the specified field in CM memory to an image file using the current settings of the *image_info* structure.

Before calling this routine, an image information structure must be allocated and the parameters for the image width, height, samples per pixel, bits per sample, and photometric interpretation must be set. If the photometric interpretation is palette, you must also supply a color map. See the description of the Image File Interface routines that set these parameters for more information.

CMSR_image_field_to_file returns non-negative integers on success and **CMSR_IMAGE_ERROR** if an error is detected.

SEE ALSO

CMSR_image_field_to_file_simple

CMSR_image_field_to_file_simple

Transfers an image from CM memory to a file without using an image information structure.

SYNTAX

C Syntax

```
#include <cm/cmtiff.h>

int
   CMSR_image_field_to_file_simple
       (*filename, src_field, src_len, append_p, x_varies_fastest_p, width, height,
       photometric_interpretation, samples_per_pixel, bits_per_sample, red,
       green, blue);

char            *filename;
CM_field_id_t   src_field;
unsigned int    src_len;
int             append_p;
int             x_varies_fastest_p;
int             width, height;
int             photometric_interpretation;
int             samples_per_pixel, bits_per_sample;
float           red[], green[], blue[];
```

Fortran Syntax

```
    INCLUDE '/usr/include/cm/cmtiff-fort.h'

    INTEGER FUNCTION CMSR_IMAGE_FIELD_TO_FILE_SIMPLE
&       (filename, src_field, src_len, append_p, x_varies_fastest_p, width, height,
&         photometric_interpretation, samples_per_pixel, bits_per_sample, red,
&        green, blue)

    CHARACTER*(*)   filename
    INTEGER         src_field
    INTEGER         src_len
    LOGICAL         append_p
    LOGICAL         x_varies_fastest_p
    INTEGER         width, height
    INTEGER         photometric_interpretation
INTEGER       samples_per_pixel, bits_per_sample
REAL          red(), green(), blue()
```

ARGUMENTS

filename The file to which the image is to be written.

src_field The field ID of the field in CM memory from which the image is to be read.

src_len The length, in bits, of the source field containing the image.

append_p A predicate indicating whether the image is appended to or overwrites *filename*. If *append_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp), the image from the array is appended to the end of the file. If *append_p* is false (.FALSE. in Fortran, NULL in C, nil in Lisp), the image from the field overwrites any images already in the file.

x_varies_fastest_p A boolean or logical specifying whether the image is stored such that the *x* or *y* coordinate varies fastest. Normally, *x_varies_fastest_p* should be set to true.

If *x_varies_fastest_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp), the image data is organized in the file so that the *x* coordinate varies fastest, that is, in column-major order. This is the case for Fortran language arrays, and for C arrays that are referenced [*y*][*x*].

If *x_varies_fastest_p* is false (.FALSE. in Fortran, NULL in C, nil in Lisp), the image data is organized so that the *y* coordinate varies fastest, that is, in row-major order. This is the case for C arrays referenced [*x*][*y*].

The *Render display routines assume that the *x* coordinate varies fastest and maps axis 0 of the image field to the *x* dimension of the display. If your image arrays are stored so that *y* varies fastest, *x_varies_fastest* should be set to FALSE.

width If *x_varies_fastest_p* is true, *width* is the length of axis 0 of the field. If *x_varies_fastest_p* is FALSE, *width* is the length of axis 1.

height If *x_varies_fastest_p* is true, *height* is the length of axis 1 of the field. If *x_varies_fastest_p* is FALSE, *height* is the length of axis 0.

photometric_interpretation

An enumerated variable specifying how the image data is to be interpreted by Image File Interface.

Possible values can be:

- `PHOTOMETRIC_MINISWHITE`
- `PHOTOMETRIC_MINISBLACK`
- `PHOTOMETRIC_RGB`
- `PHOTOMETRIC_PALETTE`

samples_per_pixel The number of color components that are maintained per pixel in the image to be stored. Currently, the number of samples supported by the Image File Interface are

- 1 for bilevel, grayscale, or palette images
- 3 (red, green, and blue) for RGB true color images
- 4 (red, green, blue, and alpha) for RGB plus alpha channel images

bits_per_sample The number of bits of color information maintained for each sample per pixel in the image. For one sample per pixel the bits per sample may be 1, 2, 4, or 8. For three or four samples per pixel the bits per sample must be 8. For the most common classes of images, the following numbers of bits per sample are supported:

- Bilevel images must be 1 bit per sample
- Grayscale images can be 4 or 8 bits per sample
- Palette color images can be 1, 2, 4, or 8 bits per sample
- RGB full-color images must be 8 bits per sample
- RGB plus alpha images must be 8 bits per sample

red, green, blue Arrays of color values specifying the red, green, and blue components of a color map. The arrays must each have 2^(*samples_per_pixel*) elements.

If the photometric interpretation is **`PHOTOMETRIC_PALETTE`**, these arrays must be specified. For palette images, the color map defined by these arrays is stored in the file with the image and used to initialize the Generic Display when the file is read back into the system.

If *photometric_interpretation* has any value other than **PHOTOMETRIC_PALETTE**, this color map is ignored.

The user is responsible for allocating memory to hold the arrays.

NOTE: If you want to load a color ınap from a generic display, you can use the Generic Display Interface routine **CMSR_display_read_color_map**.

DESCRIPTION

CMSR_image_field_to_file_simple transfers an image directly from the specified field to an image file.

CMSR_image_field_to_file_simple returns non-negative integers on success and **CMSR_IMAGE_ERROR** if an error is detected.

SEE ALSO

CMSR_image_field_to_file

CMSR_image_file_to_array

Transfers an image from an image file to a front-end array.

SYNTAX

C Syntax

```
#include <cm/cmtiff.h>

int
   CMSR_image_file_to_array
                    (*filename, image_array, image_info, image_number)

char                     *filename
CMSR_generic_pointer_t   image_array
CMSR_image_info_t        image_info
int                      image_number
```

Fortran Syntax

```
    INCLUDE '/usr/include/cm/cmtiff-fort.h'

    INTEGER FUNCTION CMSR_IMAGE_FILE_TO_ARRAY
&                     (filename, image_array, image_info, image_number)

    CHARACTER*(*)  filename
    ANY_FE_ARRAY   image_array
    INTEGER        image_info
    INTEGER        image_number
```

ARGUMENTS

filename The file from which the image is to be read.

image_array The array on the front-end computer to which the image is to be written. The image array may be of any data type that provides an appropriate number of bits for the depth of the image stored in the file.

In C the **CMSR_generic_pointer_t** data type accepts a pointer to any data type. It is used to make code portable between ANSI compilers that accept pointers to **void** and VAX compilers that would require a pointer to **char** for unspecified data types. You may need to cast this variable to the data type you use for your array to avoid compiler warnings.

image_info A **CMSR_image_info_t** data structure that is filled with specifications for the image when it is loaded from the file. If you do not wish to load this information, you can pass in NULL or zero for this field. This argument is then ignored and data describing the image is not returned.

image_number The number of the image in the TIFF file that should be displayed. TIFF files may contain multiple images. The Image File Interface numbers images starting at zero. If the number of images in the file is less than the requested number, **EOF** (–1) is returned.

DESCRIPTION

CMSR_image_file_to_array transfers an image from *filename* to the front-end array *image_array*.

When reading an image from a file to a front-end array, this routine automatically sets the *image_info* structure to describe the image.

CMSR_image_file_to_array returns non-negative integers on success, **EOF** (–1) on end of file, and **CMSR_IMAGE_ERROR** if an error is detected.

If there are not enough images in the file to fulfill a read request, **EOF** is returned.

ERRORS

An error is signalled if

the specified file cannot be opened for reading

an error occurs while reading scanlines from the file

SEE ALSO

CMSR_image_file_to_field

CMSR_image_file_to_display

CMSR_image_file_to_display

Transfers an image directly from a file to the current generic display.

SYNTAX

C Syntax

```
#include <cm/cmtiff.h>

int
  CMSR_image_file_to_display
                        (*filename, image_info, image_number)

char                *filename
CMSR_image_info_t   image_info
int                 image_number
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmtiff-fort.h'

   INTEGER FUNCTION CMSR_IMAGE_FILE_TO_DISPLAY
&                          (filename, image_info, image_number)

   CHARACTER*(*)  filename
   INTEGER        image_info
   INTEGER        image_number
```

ARGUMENTS

filename The file from which the image is to be read.

image_info A **CMSR_image_info_t** data structure that is filled with specifications for the image when it is loaded from the file. If you do not wish to load this information, you can pass in NULL or zero for this field. This argument is then ignored and data describing the image is not returned.

image_number The number of the image in the TIFF file that should be displayed. TIFF files may contain multiple images. The Image File Interface numbers images starting at zero. If the number of images in the file is less than the requested number, **EOF** (–1) is returned.

DESCRIPTION

CMSR_image_file_to_display transfers an image directly from *filename* to the currently selected Generic Display display. The display should be at least as deep as the image stored in the file.

You can read the depth of the image from the image information structure with **CMSR_image_depth**. You can read the information structure from a file before transferring the image by calling **CMSR_image_get_info**.

When reading an image from a file to a Generic Display display, this routine automatically sets the *image_info* structure to describe the image.

If the image has a color map, that color map is installed on the generic display. If the image is a grayscale image, the correct grayscale color map (based on the photometric interpretation specified with the file) is installed.

CMSR_image_file_to_display returns non-negative integers on success, **EOF** (–1) on end of file, and **CMSR_IMAGE_ERROR** if an error is detected.

If there are not enough images in the file to fulfill a read request, **EOF** is returned.

ERRORS

An error is signalled if

the specified file cannot be opened for reading

an error occurs while reading scanlines from the file

SEE ALSO

CMSR_image_file_to_field

CMSR_image_file_to_array

CMSR_image_file_to_field

Transfers an image from a TIFF image file to an image buffer field in CM memory.

SYNTAX

C Syntax

```
#include <cm/cmtiff.h>

int
   CMSR_image_file_to_field
        (*filename, dest_field, dlen, image_info, image_number,
         x_varies_fastest_p)

char               *filename;
CM_field_id_t      dest_field;
unsigned int       dlen;
CMSR_image_info_t  image_info;
int                image_number;
int                x_varies_fastest_p;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmtiff-fort.h'

   INTEGER FUNCTION CMSR_IMAGE_FILE_TO_FIELD
&       (filename, dest_field, dlen, image_info, image_number, x_varies_fastest_p)

   CHARACTER*(*)  filename
   INTEGER        dest_field
   INTEGER        d_len
   INTEGER        image_info
   INTEGER        image_number
   LOGICAL        x_varies_fastest_p
```

ARGUMENTS

filename The file from which the image is to be read.

dest_field The field ID of the field in CM memory to which the image is to be written.

dlen

The length, in bits, of the destination field that is to contain the image.

The field should be at least as deep as the image stored in the file. If it is deeper, the field is first zeroed and the image is placed in the least significant bits.

image_info

A `CMSR_image_info_t` data structure that is filled with specifications for the image when it is loaded from the file. If you do not wish to load this information, you can pass in NULL or zero for this field.

image_number

The number of the image in the TIFF file that should be displayed. TIFF files may contain multiple images. The Image File Interface numbers images starting at zero. If the number of images in the file is less than the requested number, **EOF** (–1) is returned.

x_varies_fastest_p

A boolean or logical specifying whether the image is stored in the field such that the *x* or *y* coordinate varies fastest. Normally, *x_varies_fastest_p* should be set to true.

If *x_varies_fastest_p* is true (.TRUE. in Fortran, non-NULL in C, non-nil in Lisp), the image data is organized so that the *x* coordinate varies fastest, that is, in column-major order. This is the case for Fortran language arrays, and for C arrays that are referenced [*y*][*x*].

If *x_varies_fastest_p* is false (.FALSE. in Fortran, NULL in C, nil in Lisp), the image data is organized so that the *y* coordinate varies fastest, that is, in row-major order. This is the case for C arrays referenced [*x*][*y*].

The *Render display routines assume that *x* varies fastest and map axis 0 of the image field to the *x* dimension of the display. If your image arrays are stored so that *y* varies fastest, *x_varies_fastest* should be set to FALSE.

DESCRIPTION

`CMSR_image_file_to_field` transfers an image directly from an image file to the specified CM field. The field should be at least as deep as the image stored in the file. If the field is deeper than the image data, the field is first zeroed and the image data is placed in the least significant bits.

You can read the depth of the image from the image information structure with **CMSR_image_depth**. You can read the information structure from a file before transferring the image by calling **CMSR_image_get_info**.

This routine returns non-negative integers on success, **EOF** (–1) on end of file, and **CMSR_IMAGE_ERROR** if an error is detected.

If there are not enough images in the file to fulfill a read request, **EOF** is returned.

ERRORS

An error is signalled if

the specified file cannot be opened for reading

an error occurs while reading scanlines from the file

SEE ALSO

CMSR_image_file_to_display

CMSR_image_file_to_array

cmdisplay2tiff

Copies an image from CM framebuffer to a TIFF file.

SYNTAX

UNIX Shell-Level Command

```
cmdisplay2tiff [-append] [-x offset] [-y offset] [-w width]
[-h height] [-artist name] [-host name]
[-description string] [-software name]
[-separate] [-nocomp | -lzw]
[-rowsperstrip number] filename
```

ARGUMENTS

-append Adds the current image to the end of the file.

-x *offset* Starting pixel offset from the left of the screen. Defaults to zero.

-y *offset* Starting pixel offset from the top of the screen. Defaults to zero.

-w *width* Width in pixels of the image region to transfer. Defaults to width of the screen divided by current *x* zoom factor.

-h *height* Height in pixels of the image region to transfer. Defaults to the height of the screen divided by current *y* zoom factor.

-dpi *number* Specify the number of dots per inch (in both x and y) for the image. The default is to omit resolution information from the created file. Note that applications are not required to use this information even when it is present in a TIFF file.

-artist *name* Name of the creator of the image. Defaults to user name.

-host *name* Name of the machine on which the image was created. Defaults to name of the local host.

-description *string*
Optional description to be stored with the image. Defaults to NULL.

`-software` *name* Name of the software used to created the image. Defaults to **`cmdisplay2tiff`**.

`-separate` Stores RGB images in separate planes for each color component. The default is to store images contiguously, but separate planes may compress more efficiently for some images.

`-nocomp | -lzw`

Type of compression to be used when storing the image. **`-lzw`** is the default.

`-rowsperstrip` *number*

Number of scanlines in each strip of data in the output file. By default this value is set so that the size of each strip is as close to 8K bytes as possible without going over.

filename File name in which the image is to be stored.

DESCRIPTION

`cmdisplay2tiff` copies an image from a portion of a CM framebuffer to a TIFF file. You must be attached to a CM from a sequencer that can access the desired framebuffer.

If the environment variable **`CM_DISPLAY`** is set to the name of a CM framebuffer (as described in **`CMSR_select_display_menu`**), that display is used as the source of the image.

If **`CM_DISPLAY`** is not set and more than one framebuffer is available, a menu of possible framebuffers is presented to the user.

If only one framebuffer is available, it is used as the source of the image.

NOTE: This command only transfers images from a CM framebuffer generic display. If the current generic display is an X11 display window, you cannot use `cmdisplay2tiff`.

SEE ALSO

`tiff2cmdisplay`

tiff2cmdisplay

Copies an image from a TIFF file to a Generic Display Interface display.

SYNTAX

UNIX Shell-Level Command

```
tiff2cmdisplay [-x offset] [-y offset] [-image number]
      [-v] [-nozoom] [-nowait] filename
```

ARGUMENTS

-x *offset* The pixel offset from the left edge of the screen. This field only applies to CM framebuffers, and defaults to zero.

-y *offset* The pixel offset from the top edge of the screen. This field only applies to CM framebuffers, and defaults to zero.

-nozoom This flag specifies that the framebuffer is not to be panned or zoomed.

-nowait When using an X display, **tiff2cmdisplay** waits after the image is displayed until a mouse button is pressed in the window. The **nowait** option specifies that the program is to exit without waiting for a button press.

-image *number* Image to be read from the TIFF file. The first image is number zero.

-v Verbose mode. This flag causes the contents of the image author, software, host computer, description, width, height, and depth fields to be printed.

DESCRIPTION

tiff2cmdisplay reads an image from a TIFF file and displays it on the current generic display. The display may be an X11 server display or a CM framebuffer.

If the display is a CM framebuffer, the framebuffer is panned so that the offsets specified with the **-x** and **-y** options are at the origin, and is zoomed so that the image nearly fills the screen without being clipped. If you do not explicitly specify offsets, the CM framebuffer is panned so that the image is centered.

SEE ALSO

`cmdisplay2tiff`

2.2 Image Information Structure Routines

This section describes the Image File Interface routines, which set or return fields in the image information structure.

Routines are provided that allocate and deallocate an image information structure, and fill an image information structure with the image description from a specified file:

Routines that set and read back the image parameters in an Image File Interface image information structure are grouped according to whether they set and return image attribute parameters, image environment notes, or file format parameters.

Image Attribute Parameters

Image Environment Notes

File Format Parameters

CMSR_allocate_image_info
CMSR_deallocate_image_info

Allocates (deallocates) an image information data structure.

SYNTAX

C Syntax

```
#include <cm/cmtiff.h>

CMSR_image_info_t
  CMSR_allocate_image_info ()

void
  CMSR_deallocate_image_info (image_info)

CMSR_image_info_t image_info
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmtiff-fort.h'

INTEGER FUNCTION CMSR_ALLOCATE_IMAGE_INFO ()

SUBROUTINE CMSR_DEALLOCATE_IMAGE_INFO (image_info)

INTEGER  image_info
```

ARGUMENTS

image_info	A **CMSR_image_info_t** data structure containing specifications for an image and for the format in which it is to be stored.

DESCRIPTION

CMSR_allocate_image_info allocates and returns an image information data structure. **CMSR_deallocate_image_info** deallocates a specified image information data structure.

The Image File Interface routines use the image information structure to interpret data when transferring images to or from an image file. When an image is written from the CM system to a file, the transfer routines use this information to organize the image data in the file and store it with the image data. When a stored image is read back into the CM system, these routines automatically load this information into an image information structure and use it to configure the display or array to which the image is sent.

The Image File Interface is described in more detail in Chapter 1.

SEE ALSO

`CMSR_image_get_info`

CMSR_image_get_info

Fills image information structure with image description from a specified file.

SYNTAX

C Syntax

```
#include <cm/cmtiff.h>

int
  CMSR_image_get_info  (filename, image_info, image_number) ;

char               *filename;
CMSR_image_info_t  image_info;
int                image_number;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmtiff-fort.h'

   INTEGER FUNCTION CMSR_IMAGE_GET_INFO
&                             (filename, image_info, image_number)

   CHARACTER*(*)  filename;
   INTEGER        image_info;
   INTEGER        image_number;
```

ARGUMENTS

filename The name of the image file from which to read the image information.

image_info The **CMSR_image_info_t** data structure (created with **CMSR_allocate_image_info**) that is to be filled.

image_number The number of the image in *filename* from which data should be read. TIFF files may contain multiple images. The Image File Interface numbers images starting at zero. If the number of images in the file is less than the requested number, **EOF** (–1) is returned.

DESCRIPTION

CMSR_image_get_info fills in the *image_info* structure with data describing *image_number* image from *filename.* It does not read in the image data itself.

This routine returns a non-negative integer if is is successful, **EOF** (–1) on end of file, and **CMSR_IMAGE_ERROR** if an error is detected.

If there are not enough images in the file to fulfill a read request, **EOF** is returned.

ERRORS

An error is signalled if the specified file cannot be opened for reading.

SEE ALSO

CMSR_allocate_image_info

CMSR_deallocate_image_info

CMSR_image_set_format CMSR_image_format

Sets (returns) the file format in which to store images (currently only TIFF).

SYNTAX

C Syntax

```
#include <cm/cmtiff.h>

void
   CMSR_image_set_format  (image_info, image_format);

CMSR_image_info_t     image_info;
CMSR_image_format_t   image_format;


CMSR_image_format_t
    CMSR_image_format  (image_info);

CMSR_image_info_t     image_info;
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmtiff-fort.h'

SUBROUTINE CMSR_IMAGE_SET_FORMAT(image_info, image_format)

INTEGER   image_info
INTEGER   image_format


INTEGER FUNCTION CMSR_IMAGE_FORMAT(image_info)

INTEGER   image_info
```

ARGUMENTS

image_info A **`CMSR_image_info_t`** data structure containing specifications for an image and for the format in which it is to be stored.

Image information structures are created with **`CMSR_allocate_image_info`**.

image_format The image file format in which the image is to be stored. Currently, only **`CMSR_tiff_file`** is supported.

DESCRIPTION

`CMSR_image_set_format` sets the file format in the image information structure specified by *image_info*.

I/O routines in the Image File Interface that specify *image_info* will store image data in the format specified by *image_format.*

`CMSR_image_format` returns the image format currently set in the specified image information structure.

Currently, the only valid value for *image_format* is **`CMSR_tiff_file`**.

For more information on the TIFF file format, see Chapter 1.

CMSR_image_set_width / CMSR_image_width
CMSR_image_set_height / CMSR_image_height

Sets (returns) image width (height).

SYNTAX

C Syntax

```
#include <cm/cmtiff.h>

void
   CMSR_image_set_width  (image_info, width);

void
   CMSR_image_set_height  (image_info, height);

CMSR_image_info_t image_info;
int               width;
int               height;


int
   CMSR_image_width  (image_info);

int
   CMSR_image_height  (image_info);

CMSR_image_info_t image_info;
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmtiff-fort.h'

SUBROUTINE CMSR_IMAGE_SET_WIDTH(image_info, width)

SUBROUTINE CMSR_IMAGE_SET_HEIGHT(image_info, height)

INTEGER  image_info
INTEGER  width
INTEGER  height
INTEGER FUNCTION CMSR_IMAGE_WIDTH (image_info)
INTEGER FUNCTION CMSR_IMAGE_HEIGHT (image_info)
INTEGER image_info
```

ARGUMENTS

image_info — A **CMSR_image_info_t** data structure (created with **CMSR_allocate_image_info**) containing specifications for an image and the format in which it is to be stored.

width — An integer specifying the number of pixels in the image's *x* (horizontal) dimension.

height — An integer specifying the number of pixels in the image's *y* (vertical) dimension.

NOTE: There are no defaults in the image information structure for *width* and *height*. You must make sure that these fields are set correctly before you write an image with **CMSR_image_field_to_file** or **CMSR_image_array_to_file**.

CMSR_image_display_to_file determines image information defaults from the currently selected generic display.

DESCRIPTION

CMSR_image_set_width and **CMSR_image_set_height** set the image width and height, respectively, in the image information structure specified by *image_info*.

CMSR_image_width and **CMSR_image_height** return, respectively, the current width and height, set in the image information structure specified by *image_info*.

I/O routines in the Image File Interface that specify *image_info* configure the image file, or display for image data that is (*width* x *height*).

SEE ALSO

CMSR_image_depth

CMSR_image_depth

Returns the number of bits per pixel.

SYNTAX

C Syntax

```
#include <cm/cmtiff.h>

int
  CMSR_image_depth (image_info)

CMSR_image_info_t image_info
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmtiff-fort.h'

INTEGER FUNCTION CMSR_IMAGE_DEPTH (image_info)

INTEGER image_info
```

ARGUMENTS

image_info A **CMSR_image_info_t** data structure (created with **CMSR_allocate_image_info**) containing specifications for an image and for the format in which it is to be stored.

DESCRIPTION

CMSR_image_depth returns the depth, in bits, for the image defined by *image_info.*

The depth of the image is determined by the number of samples per pixel times the number of bits per sample. This is the total length of the field or variable needed to hold the image.

SEE ALSO

`CMSR_image_set_width`

`CMSR_image_width`

`CMSR_image_set_height`

`CMSR_image_height`

CMSR_image_set_num_samples CMSR_image_num_samples

Sets (returns) the number of samples (color components).

SYNTAX

C Syntax

```
#include <cm/cmtiff.h>

void
  CMSR_image_set_num_samples(image_info, samples_per_pixel);

CMSR_image_info_t image_info;
int               samples_per_pixel;


int
  CMSR_image_num_samples (image_info);

CMSR_image_info_t image_info;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmtiff-fort.h'

   SUBROUTINE CMSR_IMAGE_SET_NUM_SAMPLES
&                                       (image_info, samples_per_pixel)

   INTEGER image_info
   INTEGER samples_per_pixel

   INTEGER FUNCTION CMSR_IMAGE_NUM_SAMPLES (image_info)
   INTEGER  image_info
```

ARGUMENTS

image_info A **CMSR_image_info_t** data structure (created with **CMSR_allocate_image_info**) containing specifications for an image and for the format in which it is to be stored.

samples_per_pixel The number of color components of image information that are maintained for each pixel in the image.

DESCRIPTION

`CMSR_image_set_num_samples` sets the number of samples per pixel in the *image_info* image information structure.

`CMSR_image_num_samples` returns the number of samples per pixel.

The number of samples is the number of color components that are maintained per pixel in the image to be stored. Currently, the number of samples supported by the Image File Interface are

1 for bilevel, grayscale, or palette images

3 (red, green, and blue) for RGB true color images

4 (red, green, blue, and alpha) for RGB plus alpha channel images

NOTE: When the number of samples per pixel is greater than one, each sample must have the same number of bits.

There is no default in the image information structure for *samples_per_pixel.* Before you write an image with **`CMSR_image_field_to_file`** or **`CMSR_image_array_to_file`**, make sure that *samples_per_pixel* is set correctly.

`CMSR_image_display_to_file` determines image information defaults from the currently selected generic display.

SEE ALSO

`CMSR_image_set_bits_per_sample`

`CMSR_image_bits_per_sample`

`CMSR_image_depth`

CMSR_image_set_bits_per_sample CMSR_image_bits_per_sample

Sets (returns) number of bits per sample.

SYNTAX

C Syntax

```
#include <cm/cmtiff.h>

void
  CMSR_image_set_bits_per_sample(image_info, bits_per_sample)

CMSR_image_info_t image_info;
int               bits_per_sample;


int
  CMSR_image_bits_per_sample (image_info);

CMSR_image_info_t image_info;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmtiff-fort.h'

   SUBROUTINE CMSR_IMAGE_SET_BITS_PER_SAMPLE
&                                    (image_info, bits_per_sample)

   INTEGER image_info
   INTEGER bits_per_sample


   INTEGER FUNCTION CMSR_IMAGE_BITS_PER_SAMPLE (image_info)

   INTEGER  image_info
```

ARGUMENTS

image_info A **CMSR_image_info_t** data structure (created with **CMSR_allocate_image_info**) containing specifications for an image and for the format in which it is to be stored.

bits_per_sample For one sample per pixel, the bits per sample may be 1, 2, 4, or 8. For three or four samples per pixel, the bits per sample must be 8.

DESCRIPTION

CMSR_image_set_bits_per_sample sets the number of bits per sample in the *image_info* image information structure.

CMSR_image_bits_per_sample returns the number of bits per sample.

The number of samples is the number of color components that are maintained per pixel in the image to be stored.

The number of bits per sample determines the number of colors that may be represented in the color map used to display the image. For example, an image with one sample per pixel and 1 bit per sample can express only two colors, 0 or 1. A one-sample image with 8 bits per sample can reference 256 colors, 0 to 255.

For the most common classes of images, the following numbers of bits per sample are supported:

Bilevel images must be 1 bit per sample

Grayscale images can be 4 or 8 bits per sample

Palette color images can be 1, 2, 4, or 8 bits per sample

RGB full-color images must be 8 bits per sample

RGB plus alpha images must be 8 bits per sample

When the number of samples per pixel is greater than one, each sample must have the same number of bits.

NOTE: There is no default in the image information structure for *bits_per_sample*. You must make sure that this field is set correctly before you write an image with **CMSR_image_field_to_file** or **CMSR_image_array_to_file**.

CMSR_image_display_to_file determines image information defaults from the currently selected generic display.

SEE ALSO

`CMSR_image_set_num_samples`

`CMSR_image_num_samples`

`CMSR_image_depth`

CMSR_image_set_photometric CMSR_image_photometric

Sets (returns) TIFF format photometric interpretation in an image information structure.

SYNTAX

C Syntax

```
#include <cm/cmtiff.h>

void
  CMSR_image_set_photometric  (image_info, photometric);

CMSR_image_info_t image_info;
int               photometric;


int
  CMSR_image_photometric  (image_info);

CMSR_image_info_t image_info;
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmtiff-fort.h'

SUBROUTINE CMSR_IMAGE_SET_PHOTOMETRIC  (image_info, photometric)

INTEGER image_info
INTEGER photometric


INTEGER FUNCTION CMSR_IMAGE_PHOTOMETRIC  (image_info)
INTEGER image_info
```

ARGUMENTS

image_info
A **`CMSR_image_info_t`** data structure containing specifications for an image and for the format in which it is to be stored.

Image information structures are created with **`CMSR_allocate_image_info`**.

photometric
Specifies how the image data is to be interpreted by Image File Interface. Possible values can be:

- **`PHOTOMETRIC_MINISWHITE`**
- **`PHOTOMETRIC_MINISBLACK`**
- **`PHOTOMETRIC_PALETTE`**
- **`PHOTOMETRIC_RGB`**

NOTE: There is no default photometric interpretation in the image information structure. You must make sure that this field is set correctly before you write an image with **`CMSR_image_field_to_file`** or **`CMSR_image_array_to_file`**.

`CMSR_image_display_to_file` determines image information defaults from the currently selected generic display.

DESCRIPTION

`CMSR_image_set_photometric` sets the photometric interpretation of the image data. The photometric interpretation specifies how the samples of color information stored for each pixel are to be interpreted. To interpret the image data correctly, a photometric interpretation field must be supplied. This tells applications whether the image is RGB, palette color, or whether the minimum pixel value is black or white.

`CMSR_image_photometric` returns the photometric interpretation of the image data.

The photometric interpretation values are interpreted as follows:

`PHOTOMETRIC_MINISWHITE`
The image data is to be mapped so that pixels with a value of 0 are rendered as white.

This photometric interpretation may be used with bilevel or grayscale images. In the case of bilevel displays, pixels with a value of 0 are displayed as white and pixels with a value of 1 are black. In the case of grayscale images, the pixel value is mapped to a range of gray intensities running from white at 0 to black at the maximum color map entry. The number of color map entries is determined by the number of bits per sample.

PHOTOMETRIC_MINISBLACK
The image data is to be mapped so that pixels with a value of 0 are rendered as black.

This photometric interpretation may be used with bilevel or grayscale images. In the case of bilevel displays, pixels with a value of 0 are displayed as black and pixels with a value of 1 are white. In the case of grayscale images, the pixel value is mapped to a range of gray intensities running from black at 0 to white at the maximum color map entry. The number of color map entries is determined by the number of bits per sample.

PHOTOMETRIC_PALETTE
The image data is to be interpreted as an index into color map. The data must have one sample per pixel, and the number of bits per sample can be 1, 2, 4, or 8.

The number of entries in the color map is 2^(bits per sample) entries. For example, an 8-bit color map has 256 entries indexed from 0 to 255; a 4-bit map has 16 entries indexed from 0 to 15.

PHOTOMETRIC_RGB
The image data is to be interpreted as RGB true-color data. The images must have three samples per pixel and 8 bits per sample. Each sample expresses a color intensity from 0 (off) to 255 (full intensity.) The first sample is used to determine the red intensity of the color, the second sample the green intensity, and the third sample the blue intensity.

CMSR_image_set_color_map
CMSR_image_color_map

Sets (returns) color map arrays in an image information structure.

SYNTAX

C Syntax

```
#include <cm/cmtiff.h>

void
  CMSR_image_set_color_map  (image_info, red, green, blue)

CMSR_image_info_t image_info
float             red[], green[], blue[];


void
  CMSR_image_color_map  (image_info, red, green, blue)

CMSR_image_info_t image_info
float             red[], green[], blue[];
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmtiff-fort.h'

SUBROUTINE CMSR_IMAGE_SET_COLOR_MAP  (image_info, red, green, blue)

INTEGER  image_info
REAL     red(), green(), blue()

SUBROUTINE CMSR_IMAGE_COLOR_MAP  (image_info, red, green, blue)

INTEGER  image_info
REAL     red(), green(), blue()
```

ARGUMENTS

image_info A **CMSR_image_info_t** data structure (created with **CMSR_allocate_image_info**) containing specifications for an image and for the format in which it is to be stored.

*red, green, bl*ue Arrays of color values specifying the red, green, and blue components of a color map. The arrays must each have 2^(*samples_per_pixel*) elements.

The user is responsible for allocating memory to hold the arrays.

DESCRIPTION

CMSR_image_set_color_map sets the color map arrays in the image information structure.

CMSR_image_color_map returns the color map arrays for palette color images.

The user is responsible for allocating memory to hold the color map. Each of the three elements must point to an area big enough to hold 2^(bits per sample) floats.

When photometric interpretation is set to indicate a palette color image, a color map must be present.

SEE ALSO

CMSR_display_read_color_map

CMSR_image_set_artist
CMSR_image_artist

Stores (returns) artist's name in an image information structure.

SYNTAX

C Syntax

```
#include <cm/cmtiff.h>

void
   CMSR_image_set_artist  (image_info, *string);

CMSR_image_info_t image_info;
char              *string;


char *
   CMSR_image_artist  (image_info);

CMSR_image_info_t image_info;
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmtiff-fort.h'

SUBROUTINE CMSR_IMAGE_SET_ARTIST  (image_info, string)

INTEGER         image_info
CHARACTER*(*)   string


CHARACTER*(*) FUNCTION CMSR_IMAGE_ARTIST  (image_info)


INTEGER   image_info
```

ARGUMENTS

image_info A **CMSR_image_info_t** data structure (created with **CMSR_allocate_image_info**) containing specifications for an image and for the format in which it is to be stored.

string A character string containing the artist's name.

DESCRIPTION

CMSR_image_set_artist sets the artist field in the image information structure.

CMSR_image_artist returns the artist field from the image information structure.

Artist defaults to the user's login name.

CMSR_image_set_date_time CMSR_image_date_time

Stores (returns) date and time in an image information structure.

SYNTAX

C Syntax

```
#include <cm/cmtiff.h>

void
   CMSR_image_set_date_time  (image_info, *string)

CMSR_image_info_t image_info
char              *string


char *
   CMSR_image_date_time  (image_info)

CMSR_image_info_t image_info
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmtiff-fort.h'

SUBROUTINE CMSR_IMAGE_SET_DATE_TIME  (image_info, string)

INTEGER         image_info
CHARACTER*(*)   string


CHARACTER*(*) CMSR_IMAGE_DATE_TIME  (image_info)

INTEGER   image_info
```

ARGUMENTS

image_info — A **CMSR_image_info_t** data structure containing specifications for an image and for the format in which it is to be stored.

Image information structures are created with **CMSR_allocate_image_info**.

string — A character string containing the date and time.

DESCRIPTION

CMSR_image_set_date_time sets the date and time field in the image information structure.

CMSR_image_date_time returns the date and time field from the image information structure.

Date and time default to current date and time on the system.

CMSR_image_set_host_computer
CMSR_image_host_computer

Stores (returns) name or type of host computer in an image information structure.

SYNTAX

C Syntax

```
#include <cm/cmtiff.h>

void
  CMSR_image_set_host_computer (image_info, *string);

CMSR_image_info_t image_info;
char              *string;


char *
  CMSR_image_host_computer (image_info);

CMSR_image_info_t image_info;
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmtiff-fort.h'

SUBROUTINE CMSR_IMAGE_SET_HOST_COMPUTER (image_info, string)

INTEGER         image_info
CHARACTER*(*)   string


CHARACTER*(*) CMSR_IMAGE_HOST_COMPUTER (image_info)

INTEGER  image_info
```

ARGUMENTS

image_info A **CMSR_image_info_t** data structure (created with **CMSR_allocate_image_info**) containing specifications for an image and for the format in which it is to be stored.

string A character string containing the information you wish to store in the *image_info* structure.

DESCRIPTION

`CMSR_image_set_host_computer` sets the host computer field in the image information structure.

`CMSR_image_host_computer` returns the host computer field from the image information structure.

Host computer defaults to the name of the local host.

CMSR_image_set_description
CMSR_image_description

Stores (returns) text description of the image in an image information structure.

SYNTAX

C Syntax

```
#include <cm/cmtiff.h>

void
  CMSR_image_set_description  (image_info, *string);

CMSR_image_info_t image_info;
char               *string;


char *
  CMSR_image_description  (image_info);

CMSR_image_info_t image_info;
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmtiff-fort.h'

SUBROUTINE CMSR_IMAGE_SET_DESCRIPTION  (image_info, string)

INTEGER         image_info
CHARACTER*(*)   string


CHARACTER*(*)  CMSR_IMAGE_DESCRIPTION  (image_info)

INTEGER   image_info
```

ARGUMENTS

image_info A `CMSR_image_info_t` data structure (created with `CMSR_allocate_image_info`) containing specifications for an image and for the format in which it is to be stored.

string	A character string containing the information you wish to store in the *image_info* structure.

DESCRIPTION

`CMSR_image_set_description` sets the description field in the image information structure.

`CMSR_image_description` returns a description field from the image information structure.

The description field defaults to NULL.

CMSR_image_set_software CMSR_image_software

Stores (returns) name of software used to create the image in an image information structure.

SYNTAX

C Syntax

```
#include <cm/cmtiff.h>

void
   CMSR_image_set_software (image_info, *string);

CMSR_image_info_t image_info;
char              *string;


char *
   CMSR_image_software (image_info);

CMSR_image_info_t image_info;
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmtiff-fort.h'

SUBROUTINE CMSR_IMAGE_SET_SOFTWARE (image_info, string)

INTEGER        image_info
CHARACTER*(*)  string


CHARACTER*(*) CMSR_IMAGE_SOFTWARE (image_info)

INTEGER  image_info
```

ARGUMENTS

image_info	A **CMSR_image_info_t** data structure (created with **CMSR_allocate_image_info**) containing specifications for an image and for the format in which it is to be stored.
string	A character string containing the information you wish to store in the *image_info* structure.

DESCRIPTION

CMSR_image_set_software sets the software field in the image information structure.

CMSR_image_software returns a software field from the image information structure.

The software field defaults to NULL.

CMSR_image_set_compression CMSR_image_compression

Sets (returns) the data compression method to be used.

SYNTAX

C Syntax

```
#include <cm/cmtiff.h>

void
  CMSR_image_set_compression (image_info, compression)

CMSR_image_info_t image_info;
int               compression;


int
  CMSR_image_compression (image_info);

CMSR_image_info_t image_info;
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmtiff-fort.h'

SUBROUTINE CMSR_IMAGE_SET_COMPRESSION (image_info, compression)

INTEGER image_info
INTEGER compression


INTEGER FUNCTION CMSR_IMAGE_COMPRESSION (image_info)

INTEGER image_info
```

ARGUMENTS

image_info A **CMSR_image_info_t** data structure containing specifications for an image and for the format in which it is to be stored.

Image information structures are created with **CMSR_allocate_image_info**.

compression

The value may be one of:

- **COMPRESSION_LZW**
- **COMPRESSION_NONE**

The default is **COMPRESSION_LZW**.

DESCRIPTION

CMSR_image_set_compression sets the TIFF compression scheme in the specified *image_info* structure.

The default is the general compression strategy **LZW.LZW** uses the Lempel–Ziv and Welch algorithm for data compression. This is a general purpose compression algorithm that works well on any type of image. The algorithm is described in detail in Appendix F of the TIFF (Tagged Image File Format) 5.0 specification as published by Aldus Corporation and Microsoft Corporation.

Most other TIFF readers accept the **LZW** compression strategy. However, if you are transferring the image to another graphics environment, check on the compression strategies supported by the software you will be using there. If you do not know what compression methods are supported, we recommend no compression (**NONE**). Images stored with no compression should be acceptable to nearly all TIFF readers.

CMSR_image_compression returns the compression scheme of the current image.

CMSR_image_set_planar_config CMSR_image_planar_config

Sets (returns) the configuration method to be used to store planes of color information.

SYNTAX

C Syntax

```
#include <cm/cmtiff.h>

void
  CMSR_image_set_planar_config (image_info, configuration);

CMSR_image_info_t image_info;
int               configuration;


int
  CMSR_image_planar_config (image_info);

CMSR_image_info_t image_info;
```

Fortran Syntax

```
INCLUDE '/usr/include/cm/cmtiff-fort.h'

SUBROUTINE CMSR_IMAGE_SET_PLANAR_CONFIG
&                                        (image_info, configuration)

INTEGER image_info
INTEGER configuration


INTEGER FUNCTION CMSR_IMAGE_PLANAR_CONFIG (image_info)

INTEGER image_info
```

ARGUMENTS

image_info A **CMSR_image_info_t** data structure containing specifications for an image and for the format in which it is to be stored.

Image information structures are created with **`CMSR_allocate_image_info`**.

configuration The value can be:

- **`PLANARCONFIG_CONTIG`**
- **`PLANARCONFIG_SEPARATE`**

This defaults to **`PLANARCONFIG_CONTIG`**.

DESCRIPTION

`CMSR_image_set_planar_config` sets the planar configuration of the current image.

`CMSR_image_planar_config` returns the planar configuration of the current image.

This field is only relevant for images with more than one sample per pixel. It determines whether the image color components are stored as a contiguous array or as separate planes.

The configuration values are interpreted as follows:

`PLANARCONFIG_CONTIG`
Store the color components in a single contiguous array. For example, an RGB image stored contiguously has the samples interleaved: RGBRGBRGB...

Contiguous arrays must be (*samples_per_pixel* x *width*) columns by (*height*) rows. The array must be *bits_per_sample* bits deep.

`PLANARCONFIG_SEPARATE`
Store each color component in a separate plane. An RGB image stored separately would have one plane for the red component, another for the green component, and a third for the blue component.

Separate arrays must be (*samples_per_pixel*) planes by (*width*) columns by (*height*) rows. The arrays must be *bits_per_sample* bits deep.

Your choice of configuration methods will depend on whether you will be porting the image to other software and what compression strategy you wish to use. Many software packages will only accept contiguous images because each pixel's values are available in sequence. However, many compression strategies work more efficiently when the color samples are stored in separate planes.

CMSR_image_set_rows_per_strip CMSR_image_rows_per_strip

Sets (returns) the number of scanlines stored in each strip of data in the TIFF file.

SYNTAX

C Syntax

```
#include <cm/cmtiff.h>

void
  CMSR_image_set_rows_per_strip  (image_info, rows_per_strip);

CMSR_image_info_t image_info;
int               rows_per_strip;


int
  CMSR_image_rows_per_strip  (image_info);

CMSR_image_info_t image_info;
```

Fortran Syntax

```
   INCLUDE '/usr/include/cm/cmtiff-fort.h'

   SUBROUTINE CMSR_IMAGE_SET_ROWS_PER_STRIP
&                                          (image_info, rows_per_strip)

   INTEGER image_info
   INTEGER rows_per_strip

   INTEGER FUNCTION CMSR_IMAGE_ROWS_PER_STRIP  (image_info)

   INTEGER image_info
```

ARGUMENTS

image_info A **CMSR_image_info_t** data structure containing specifications for an image and for the format in which it is to be stored.

Image information structures are created with **CMSR_allocate_image_info**.

rows_per_strip The number of rows (scanlines) of pixel data to be stored in each strip of data in the TIFF file.

DESCRIPTION

CMSR_image_set_rows_per_strip sets the number of scanlines in each strip of data in the TIFF file.

CMSR_image_rows_per_strip returns the number of scanlines in each strip of data in the TIFF file.

This parameter allows you to control the amount of data that will be sent with each read from the file. This should be adjusted to allow efficient buffering on your system. If not specified, the Image File Interface sets the parameter to result in about 8K of data being buffered at a time.

Alphabetical Index of Routines

This index lists the Image File Interface routines alphabetically.

Keyword Index of Routines

This index lists the Image File Interface routines sorted by the key words that appear in their names.