XIL Reference Manual

Sun Microsystems, Inc.
2550 Garcia Avenue
Mountain View, CA 94043
U.S.A.

A Sun Microsystems, Inc. Business

NAME

intro – introduction to the XIL library

DESCRIPTION

These XIL reference manual pages describe the syntax for using the functions contained in the XIL Imaging Library. The library follows a standard "operator" imaging model. References to images (image handles) are passed to operators, which act on the image data. Each operator allows you to specify one or more source images and a single destination image, along with the parameters necessary to perform the operation. General information on how the library handles certain concepts is provided as follows:

Topic	*Reference Manual Page*
Images	**xil_create**(3)
Regions of interest	**xil_roi_create**(3)
Color spaces	**xil_set_colorspace**(3)
Origins	**xil_get_origin**(3)
Kernels	**xil_kernel_create**(3)
Overview on compressors	**xil_compress**(3)
Specific information on XIL compressors	Consult the man page for the specific compressor.

ERRORS

Error handling is asynchronous in the XIL library. Because operations are deferred to optimize groups of operations, errors may not be reported after the function call that produces the error is made in the application program. Instead, the error is reported when the set of operations that contains the error is executed.

The XIL library uses the following categories of errors:

User (or usage) errors
CIS data errors
Resource errors
Configuration errors
System errors
Arithmetic errors
Other errors

User Errors

Errors in this category are generated when a user passes invalid parameters to XIL functions or uses the library incorrectly in some other way.

CIS Data Errors

These errors occur when a bitstream does not conform to the specification of the specified compression type.

Resource Errors

The primary finite resource that the XIL library depends on is memory. If the XIL library runs out of memory, an error message to that effect will be generated. Then, depending on what the library was doing when the request for more memory occurred, a number of secondary error messages may also be generated, indicating the failure of the library to create objects, perform operations, or complete various other tasks. Some of these errors may be System errors (see below).

Configuration Errors

Errors in this category can occur if the XIL library is improperly installed, or a necessary environment variable is not set.

System Errors

XIL performs a number of internal checks on its operation. A failure of one of these checks is called an internal error. Internal errors should not occur. If such errors do occur (in the absence of an out-of-memory error), contact customer support and give as much information as possible about the error and the situation that caused it.

Arithmetic Errors

These errors occur when XIL detects an arithmetic erro in a program (for example, dividing by zero).

For More Information

Appendix B of the *XIL Programmer's Guide* provides a list of error messages by number. It also lists which XIL functions may generate a given error message. This section lists all the functions in the XIL Imaging Library. If the function does not appear on a man page bearing its name (in other words, it is grouped with other functions on a man page bearing the name of one of those other functions), then the function whose page it appears on is printed in parentheses beneath it.

Name *(Appears on Page)*	*Description*
Cell(3)	Cell compressor / decompressor for compressed image sequences (CISs)
CellB(3)	XIL driver for CellB video compression / decompression
faxG3(3)	CCITT Group 3 compressor for CISs
faxG4 (3) (**faxG3**(3))	CCITT Group 4 compressor for CISs
H261(3)	H.261 decompressor for CISs
Jpeg(3)	JPEG compressor / decompressor for CISs
JpegLL(3)	JPEG Lossless compressor / decompressor for CISs
Mpeg1(3)	MPEG decompressor for CISs

PhotoCD(3)	Reader for Kodak Photo CD(tm) format
xgl_to_xil (3) (**xil_to_xgl**(3))	converts XGL memory / display rasters to XIL memory / display images
xil_absolute(3)	finds the absolute value of pixels of an image
xil_add(3)	adds two images
xil_add_const (3) (**xil_add**(3))	adds a constant to each band of an image
xil_affine(3)	affine-transforms an image
xil_and(3)	bitwise logical AND operation
xil_and_const (3) (**xil_and**(3))	bitwise logical AND operation with constants
xil_band_combine(3)	interband linear combination operation
xil_black_generation(3)	adjusts amount of black in a CMYK image
xil_blend(3)	blends two images according to an alpha image
xil_call_next_error_handler (3) (**xil_install_error_handler**(3))	allows error handler further down the chain to handle the error
xil_cast(3)	casts an image from one data type into another
xil_choose_colormap(3)	chooses a reasonable colormap
xil_cis_attempt_recovery(3)	attempts recovery after an error occurs in a CIS
xil_cis_create(3)	creates a new CIS

xil_cis_destroy(3)	destroys a CIS
xil_cis_flush(3)	completes pending operations for a CIS
xil_cis_get_attribute(3)	gets a compressor attribute
xil_cis_get_autorecover(3)	indicates whether a decompressor will recover automatically from a recoverable datastream error
xil_cis_get_bits_ptr(3)	gets a pointer to compressed data
xil_cis_get_by_name(3)	returns a handle to the CIS object with the specified name
xil_cis_get_compression_type(3)	returns the name of the type of the compressor
xil_cis_get_compressor(3)	returns the name of a specific compressor
xil_cis_get_input_type(3)	returns the type of image that a CIS will accept for compression
xil_cis_get_keep_frames (3) (**xil_cis_get_max_frames**(3))	gets maximum number of previously decompressed frames in buffer
xil_cis_get_max_frames(3)	gets maximum number of compressed frames in buffer
xil_cis_get_name (3) (**xil_cis_get_by_name**(3))	returns a copy of the specified CIS object’s name
xil_cis_get_output_type(3)	returns the XilImageType produced by a compressor
xil_cis_get_random_access(3)	shows whether a compressor supports random accessing of a CIS
xil_cis_get_read_frame (3) (**xil_cis_get_start_frame**(3))	returns index to the current frame

xil_cis_get_read_invalid(3)	determines whether a CIS is able to be decompressed
xil_cis_get_start_frame(3)	returns index to the first compressed image in the CIS
xil_cis_get_write_frame (3) (**xil_cis_get_start_frame**(3))	returns index to the last frame +1 of the CIS
xil_cis_get_write_invalid(3)	determines whether a CIS is able to continue to be compressed
xil_cis_has_data(3)	returns the number of bytes from the current frame to the end of the CIS
xil_cis_has_frame (3) (**xil_cis_has_data**(3))	returns TRUE if a complete frame exists at the read frame position
xil_cis_number_of_frames (3) (**xil_cis_has_data**(3))	determines number of complete unread frames of compressed data in the CIS
xil_cis_put_bits(3)	puts compressed data into a CIS
xil_cis_put_bits_ptr (3) (**xil_cis_put_bits**(3))	supplies a pointer to compressed data to a CIS
xil_cis_reset(3)	clears data in a CIS
xil_cis_seek(3)	finds a given frame of compressed data in a CIS
xil_cis_set_attribute (3) (**xil_cis_get_attribute**(3))	sets a compressor attribute
xil_cis_set_autorecover (3) (**xil_cis_get_autorecover**(3))	sets permission to attempt recovery if autorecoverable bitstream errors occur
xil_cis_set_keep_frames (3) (**xil_cis_get_max_frames**(3))	sets the number of frames prior to the current read frame to be kept in the buffer

xil_cis_set_max_frames (3) (**xil_cis_get_max_frames**(3))	sets the maximum number of frames or images in the buffer
xil_cis_set_name (3) (**xil_cis_get_by_name**(3))	sets the name of the specified CIS object to the one provided
xil_cis_sync(3)	forces any outstanding call to **xil_compress**(3) to complete when it would otherwise have been deferred
xil_close (3) (**xil_open**(3))	ends an XIL session
xil_color_convert(3)	converts an image from one color space to another
xil_colorcube_create(3)	creates a lookup table that represents a colorcube
xil_colorspace_get_by_name(3)	gets a color space object by its name
xil_compress(3)	compresses an image into a CIS
xil_convolve(3)	convolves an image with a specified kernel
xil_copy(3)	copies an image
xil_copy_pattern(3)	replicates the source image into the destination image
xil_copy_with_planemask(3)	use a plane mask to copy a source image into a destination image
xil_create(3)	creates an image
xil_create_child(3)	creates a child image
xil_create_copy(3)	creates a new image with a copy of the source's data
xil_create_from_device (3) (**xil_create_from_window**(3))	creates an image associated with the specified device

xil_create_from_type(3)	creates an image from an XilImageType object
xil_create_from_window(3)	creates an image associated with the specified X window
xil_decompress(3)	decompresses a CIS
xil_default_error_handler (3) (**xil_install_error_handler**(3))	prints error messages to the standard error output
xil_destroy(3)	destroys an image
xil_device_create(3)	creates a device object
xil_device_set_value(3)	stores device-initialization values in a device object
xil_device_destroy (3) (**xil_device_create**(3))	destroys a device object
xil_dilate (3) (**xil_erode**(3))	dilates an image
xil_dithermask_create(3)	creates a dither mask
xil_dithermask_create_copy (3) (**xil_dithermask_create**(3))	creates and returns a copy of the specified dither mask
xil_dithermask_destroy (3) (**xil_dithermask_create**(3))	destroys the specified dither mask
xil_dithermask_get_by_name(3)	returns a handle to the dither mask with the specified name
xil_dithermask_get_height(3)	gets the height of the specified dither mask
xil_dithermask_get_name (3) (**xil_dithermask_get_by_name**(3))	returns a copy of the specified dither mask's name
xil_dithermask_get_nbands (3) (**xil_dithermask_get_height**(3))	gets the number of bands in the specified dither mask

xil_dithermask_get_width (3) (**xil_dithermask_get_height**(3))	gets the width of the specified dither mask
xil_dithermask_set_name (3) (**xil_dithermask_get_by_name**(3))	sets the name of the specified dither mask to the one provided
xil_divide(3)	divides one image by another
xil_divide_by_const (3) (**xil_divide**(3))	divides a constant into each band of an image
xil_divide_into_const (3) (**xil_divide**(3))	divides each band of an image into constants
xil_erode(3)	erodes an image
xil_edge_detection(3)	detects edges within an image
xil_error_diffusion(3)	converts an image into a single-band image with a lookup table by error-diffusion dithering
xil_error_get_category (3) (**xil_error_get_string**(3))	returns the general category of the error
xil_error_get_category_string (3) (**xil_error_get_string**(3))	returns a character string that identifies the error category
xil_error_get_id (3) (**xil_error_get_string**(3))	returns a character string that uniquely identifies the error
xil_error_get_location (3) (**xil_error_get_string**(3))	returns an encrypted error location code
xil_error_get_object (3) (**xil_error_get_string**(3))	returns the XIL object that an error occurred on
xil_error_get_primary (3) (**xil_error_get_string**(3))	returns TRUE if the currently reported error is the primary cause of the error
xil_error_get_string(3)	returns an error string in the currently configured language

xil_export(3)	exports an image from XIL to application space
xil_extrema(3)	finds minimum and maximum values of an image
xil_fill(3)	performs boundary fill from a specified start point in an image
xil_get_attribute(3)	gets the values of client attributes of images
xil_get_by_name(3)	returns a handle to the image with the specified name
xil_get_child_offsets(3)	gets the values of the offsets into a parent image
xil_get_datatype(3)	gets an image's data type
xil_get_device_attribute(3)	gets the values of attributes of device images
xil_get_exported (3) (**xil_export**(3))	gets the export status of an image
xil_get_height (3) (**xil_get_width**(3))	gets the height of an image
xil_get_imagetype(3)	gets the type of an image
xil_get_info(3)	gets information about the parameters of an image
xil_get_memory_storage(3)	gets an image's memory storage
xil_get_name (3) (**xil_get_by_name**(3))	returns a copy of the specified image's name
xil_get_nbands (3) (**xil_get_width**(3))	gets the number of bands in an image
xil_get_origin(3)	gets the coordinates of the origin of an image

xil_get_origin_x (3) (**xil_get_origin**(3))	gets the x coordinate of the origin of an image
xil_get_origin_y (3) (**xil_get_origin**(3))	gets the y coordinate of the origin of an image
xil_get_parent(3)	gets a parent image
xil_get_pixel (3) (**xil_set_pixel**(3))	gets the value of a single pixel in an image
xil_get_readable(3)	returns TRUE if an image can be used as a source
xil_get_roi(3)	gets the region of interest (ROI) attached to an image
xil_get_size (3) (**xil_get_width**(3))	gets the size of an image
xil_get_synchronize (3) (**xil_sync**(3))	returns status of synchronization of an image
xil_get_width(3)	gets the width of an image
xil_get_writable (3) (**xil_get_readable**(3))	returns TRUE if an image can be used as a destination
xil_histogram(3)	generates histogram data from an image
xil_histogram_create(3)	creates a histogram object
xil_histogram_destroy (3) (**xil_histogram_create**(3))	destroys a histogram object
xil_histogram_get_by_name(3)	returns a handle to the histogram object with the specified name
xil_histogram_get_info (3) (**xil_histogram_get_nbands**(3))	gets values of histogram attributes

xil_histogram_get_limits (3) (**xil_histogram_get_nbands**(3))	gets values of arrays that represent values of the first and last bin in each band
xil_histogram_get_name (3) (**xil_histogram_get_by_name**(3))	returns a copy of the specified histogram object's name
xil_histogram_get_nbands(3)	gets the number of bands represented by the histogram
xil_histogram_get_nbins (3) (**xil_histogram_get_nbands**(3))	gets the array of values representing the number of histogram bins for each band
xil_histogram_get_values (3) (**xil_histogram_get_nbands**(3))	gets the array of values for the data attribute
xil_histogram_set_name (3) (**xil_histogram_get_by_name**(3))	sets the name of the specified histogram object to the one provided
xil_imagetype_get_by_name(3)	returns a handle to the image type object with the specified name
xil_imagetype_get_datatype(3)	gets an image type object's data type
xil_imagetype_get_height (3) (**xil_imagetype_get_width**(3))	gets the height of an image type object
xil_imagetype_get_info(3)	gets information about the parameters of an image type object
xil_imagetype_get_name (3) (**xil_imagetype_get_by_name**(3))	returns a copy of the specified image type object's name
xil_imagetype_get_nbands (3) (**xil_imagetype_get_width**(3))	gets the number of bands of an image type object
xil_imagetype_get_size (3) (**xil_imagetype_get_width**(3))	gets the size of an image type object
xil_imagetype_get_width(3)	gets the width of an image type object
xil_imagetype_set_name (3) (**xil_imagetype_get_by_name**(3))	sets the name of the specified image type object to the one provided

xil_import (3) (**xil_export**(3))	imports an image from application space to XIL space
xil_install_error_handler(3)	installs a customized error handler
xil_interpolation_table_create(3)	creates an interpolation table object
xil_interpolation_table_destroy (3) (**xil_interpolation_table_create**(3))	destroys an interpolation table object
xil_interpolation_table_get_data(3)	gets the data of an interpolation table object
xil_interpolation_table_get_kernel_size(3)	gets the kernel size of the subsample kernels in an interpolation table object
xil_interpolation_table_get_subsamples(3)	gets the number of subsamples in an interpolation table object
xil_kernel_create(3)	creates a kernel
xil_kernel_create_copy (3) (**xil_kernel_create**(3))	creates and returns a copy of the specified kernel
xil_kernel_destroy (3) (**xil_kernel_create**(3))	destroys the specified kernel
xil_kernel_get_by_name(3)	returns a handle to the kernel object with the specified name
xil_kernel_get_height(3)	gets the height of a kernel
xil_kernel_get_key_x (3) (**xil_kernel_get_height**(3))	gets the x coordinate of the key value of a kernel
xil_kernel_get_key_y (3) (**xil_kernel_get_height**(3))	gets the y coordinate of the key value of a kernel
xil_kernel_get_name (3) (**xil_kernel_get_by_name**(3))	returns a copy of the specified kernel object's name
xil_kernel_get_width (3) (**xil_kernel_get_height**(3))	gets the width of a kernel

xil_kernel_set_name (3) (**xil_kernel_get_by_name**(3))	sets the name of the specified kernel object to the one provided
xil_lookup(3)	passes an image through a lookup table
xil_lookup_convert(3)	calculates a lookup table that converts between source and destination lookup tables
xil_lookup_create(3)	creates a single lookup table
xil_lookup_create_combined(3)	creates a combined lookup table
xil_lookup_create_copy (3) (**xil_lookup_create**(3))	creates and returns a copy of the specified lookup table
xil_lookup_destroy (3) (**xil_lookup_create**(3))	destroys a lookup table
xil_lookup_get_band_lookup(3)	gets a single lookup table out of a combined lookup
xil_lookup_get_by_name(3)	returns a handle to the lookup table with the specified name
xil_lookup_get_colorcube (3) (**xil_colorcube_create**(3))	returns TRUE if a lookup table is formatted as a colorcube
xil_lookup_get_colorcube_info (3) (**xil_colorcube_create**(3))	returns formatting information about a lookup table used as a colorcube
xil_lookup_get_input_datatype(3)	gets the data type of the input to a lookup table
xil_lookup_get_name (3) (**xil_lookup_get_by_name**(3))	returns a copy of the specified lookup table's name
xil_lookup_get_num_entries (3) (**xil_lookup_get_input_datatype**(3))	gets the number of entries in a lookup table
xil_lookup_get_offset (3) (**xil_lookup_get_input_datatype**(3))	gets the offset value of a lookup table

xil_lookup_get_input_nbands (3) (**xil_lookup_get_output_nbands**(3))	gets the number of bands in the input from a lookup table
xil_lookup_get_output_datatype (3) (**xil_lookup_get_input_datatype**(3))	gets of data type of the output from a lookup table
xil_lookup_get_output_nbands (3) (**xil_lookup_get_input_datatype**(3))	gets the number of bands in the output from a lookup table
xil_lookup_get_values (3) (**xil_lookup_set_values**(3))	gets the values in a lookup table
xil_lookup_get_version(3)	gets the unique version number of a lookup table
xil_lookup_set_name (3) (**xil_lookup_get_by_name**(3))	sets the name of the specified lookup table to the one provided
xil_lookup_set_offset (3) (**xil_lookup_get_input_datatype**(3))	sets the offset value of a lookup table
xil_lookup_set_values(3)	sets the values in a lookup table
xil_max(3)	finds the larger of pixels in two images
xil_min(3)	finds the lesser of pixels in two images
xil_multiply(3)	multiplies two images
xil_multiply_const (3) (**xil_multiply**(3))	multiplies each band of an image by a floating point constant
xil_nearest_color(3)	converts an image into a single-band image by mapping pixels to the nearest entries in a lookup table
xil_not(3)	bitwise logical NOT operation
xil_object_get_error_string (3) (**xil_error_get_string**(3))	creates a string with additional information about the object involved in the error

xil_object_get_type (3) (**xil_error_get_string**(3))	returns the XilObjectType of an object
xil_open(3)	opens the XIL library for use
xil_or(3)	bitwise logical OR operation
xil_or_const (3) (**xil_or**(3))	bitwise logical OR operation with constants
xil_ordered_dither(3)	uses ordered dithering to convert an image into a single-band image with a lookup table
xil_paint(3)	blends portions of an image with a single color using a 2-D brush
xil_remove_error_handler (3) (**xil_install_error_handler**(3))	removes an error function from the error handler chain
xil_rescale(3)	rescales an image
xil_roi_add_image(3)	adds a binary image to an ROI
xil_roi_add_rect(3)	adds a rectangle to an ROI
xil_roi_add_region(3)	adds an X region to an ROI
xil_roi_create(3)	creates an ROI
xil_roi_create_copy (3) (**xil_roi_create**(3))	creates and returns a copy of an ROI
xil_roi_destroy (3) (**xil_roi_create**(3))	destroys an ROI
xil_roi_get_as_image(3)	gets an image version of an ROI
xil_roi_get_as_region(3)	returns a handle to an X region
xil_roi_get_by_name(3)	returns a handle to the ROI with the specified name

xil_roi_get_name (3) (**xil_roi_get_by_name**(3))	returns a copy of the specified ROI's name
xil_roi_intersect(3)	finds the intersection of two ROIs
xil_roi_set_name (3) (**xil_roi_get_by_name**(3))	sets the name of the specified ROI to the one provided
xil_roi_subtract_rect(3)	subtracts a rectangle from an ROI
xil_roi_translate(3)	translates an ROI
xil_roi_unite(3)	finds the union of two ROIs
xil_rotate(3)	rotates an image
xil_scale(3)	scales an image
xil_sel_create(3)	creates a structuring element (SEL)
xil_sel_create_copy (3) (**xil_sel_create**(3))	creates and returns a copy of a SEL
xil_sel_destroy (3) (**xil_sel_create**(3))	destroys a SEL
xil_sel_get_by_name(3)	returns a handle to the SEL with the specified name
xil_sel_get_height(3)	gets the height of a SEL
xil_sel_get_key_x (3) (**xil_sel_get_height**(3))	gets the x coordinate of the key value of a SEL
xil_sel_get_key_y (3) (**xil_sel_get_height**(3))	gets the y coordinate of the key value of a SEL
xil_sel_get_name (3) (**xil_sel_get_by_name**(3))	returns a copy of the specified SEL's name
xil_sel_get_width (3) (**xil_sel_get_height**(3))	gets the width of a SEL

xil_sel_set_name (3) (**xil_sel_get_by_name**(3))	sets the name of the specified SEL to the one provided
xil_set_attribute (3) (**xil_get_attribute**(3))	sets the values of client attributes of images
xil_set_colorspace(3)	sets an image’s color space
xil_set_device_attribute (3) (**xil_get_device_attribute**(3))	sets the values of attributes of device images
xil_set_memory_storage (3) (**xil_get_memory_storage**(3))	sets an exported image’s memory storage
xil_set_name (3) (**xil_get_by_name**(3))	sets the name of the specified image to the one provided
xil_set_origin (3) (**xil_get_origin**(3))	sets the coordinates of the origin of an image
xil_set_pixel(3)	sets the value of a single pixel in an image
xil_set_roi (3) (**xil_get_roi**(3))	sets an image’s ROI
xil_set_synchronize (3) (**xil_sync**(3))	sets synchronization of an image
xil_set_value(3)	sets pixels of an image to constant values
xil_soft_fill(3)	performs a soft fill from a specified starting point in an image
xil_squeeze_range(3)	produces a lookup table that will map an image into contiguous entries
xil_state_get_interpolation_tables(3)	gets interpolation tables from the XilSystemState object
xil_state_set_interpolation_tables (3) (**xil_state_get_interpolation_tables**(3))	sets interpolation tables on the XilSystemState object

xil_state_get_show_action(3)	gets the current value of the SHOW_ACTION attribute of a system-state object
xil_state_get_synchronize (3) (**xil_sync**(3))	returns synchronization status of an XIL State
xil_state_set_show_action (3) (**xil_state_get_show_action**(3))	sets the current value of the SHOW_ACTION attribute of a system-state object
xil_state_set_synchronize (3) (**xil_sync**(3))	sets synchronization status for an XIL State
xil_subsample_adaptive(3)	adaptively subsamples an image
xil_subsample_binary_to_gray(3)	subsamples a binary image and produces a grayscale image
xil_subtract(3)	subtracts one image from another
xil_subtract_const (3) (**xil_subtract**(3))	subtracts a constant from each band of an image
xil_subtract_from_const (3) (**xil_subtract**(3))	subtracts each band of an image from a constant
xil_sync(3)	forces computation of the value of an image when it would have otherwise been deferred
xil_tablewarp(3)	warps an image in both the horizontal and vertical directions
xil_tablewarp_horizontal (3) (**xil_tablewarp**(3))	warps an image in the horizontal direction
xil_tablewarp_vertical (3) (**xil_tablewarp**(3))	warps an image in the vertical direction
xil_threshold(3)	sets value of image pixel bands within a specified range

xil_to_xgl(3)	converts XIL memory / display images to XGL memory / display rasters
xil_toss(3)	throws away the contents of an image without destroying it
xil_translate(3)	translates an image
xil_transpose(3)	rotates or transposes an image
xil_xor(3)	bitwise logical XOR operation
xil_xor_const (3) (**xil_xor**(3))	bitwise logical XOR operation with constants

NAME

Cell – Cell compressor / decompressor for compressed image sequences

DESCRIPTION

The Cell image compression technology, which was developed by Sun, has been optimized for rapid decompression and display on simple hardware. Cell compression is able to achieve reasonable display quality on indexed color frame buffers. The initial focus of the Cell technology is for Sun-to-Sun communications, where the benefits of fast decode performance outweigh the benefits of standards.

The Cell encoding process transforms individual video frames into a bytestream that can be displayed with the Cell decompressor. In the first step of the encoding process, the synthetic (or filtered) video images are analyzed to produce an appropriate colormap to represent the frames to be encoded. This step allows the specification of the colormap size, in order to leave colors unused. This enhances cooperation with the window manager and other applications. Cell also provides a dynamic colormap strategy in which a new colormap is generated after each frame is compressed. This map can be used in subsequent frames.

Choosing a Colormap

The compressor chooses which colormap to encode the current image in one of three ways. If Adaptive Colormap Selection (ACS) is enabled, and a new colormap has not been associated with the compressor since the last call to **xil_compress**(3), the adapted colormap is used. When ACS is disabled, the compressor always uses the colormap given by the COMPRESSOR_COLORMAP attribute, if it has been set. If the compressor does not have a colormap, either via the COMPRESSOR_COLORMAP attribute or ACS, the compressor calls **xil_choose_colormap**(3) to generate an optimal colormap for the image. To reset ACS, give the compressor a new colormap via the COMPRESSOR_COLORMAP attribute.

Image Types

The Cell compressor and decompressor, respectively, accept and produce 3-band images in RGB color space. The width and height of the images must be divisible by 4.

Creating a Cell CIS

To compress a compressed image sequence (CIS) with the XIL Cell compressor, specify "Cell" for the *compressorname* argument in **xil_cis_create**(3).

Getting and Setting Cell Attributes

Use **xil_cis_get_attribute**(3) and **xil_cis_set_attribute**(3) to get and set Cell CIS attributes. These attributes are described in the following sections. Refer to the example section for additional information.

Cell Compression Attributes

The following paragraphs describe the Cell CIS attributes available with the XIL library. All structures and enumerations are defined via **xil.h.** Note that some attributes are "set-only" and others are "get-only." This is noted under the *Access* heading for each attribute.

Note that if you are setting an attribute and that attribute is a structure, you must pass the address of that structure. If you are getting an attribute, you always pass its address.

ENCODING_TYPE

Description	Specifies encoding algorithm
Access	get and set
Type	typedef enum { BTC, DITHER } XilCellEncodingType;
Values	*DITHER:* Use the dither encoding technique, which chooses two colors and a mask that produces the least amount of error when dithered across the 4x4 region. By selecting dither encoding, Adaptive Colormap Selection (ACS) is disabled. The current value of the COLORMAP_ADAPTION attribute is ignored. *BTC:* Use Block Truncation Coding to selection the two colors and the mask. This is much faster than dither encoding and produces good results.
Default	*BTC*

TEMPORAL_FILTERING

Description	Turns on or off a form of temporal filtering that helps with compression interframe encoding.
Access	get and set
Type	Xil_boolean
Values	*TRUE:* Filtering turned on *FALSE:* Filtering turned off
Default	TRUE

COMPRESSOR_COLORMAP

Description	Associates a colormap with the compressor for encoding images.
Access	set-only
Type	XilLookup
Default	*NULL*

COLORMAP_ADAPTION

Description	Enables or disables Adaptive Colormap Selection (ACS). ACS selects a colormap for the next image that is visibly indistinguishable from the colormap used with the current image. Thus, ACS continually adapts the colormap so it does not change enough between two images to cause colormap flashing when the bytestream is decoded.

	ACS detects when an adapted colormap has too much error and encodes new colormaps until the colormaps closely match the optimal colormap for the image. So, when ACS is enabled, every frame may have a new colormap associated with it.
Access	get and set
Type	Xil_boolean
Values	*TRUE/FALSE*
Default	*TRUE*
Notes	ACS is disabled when using dither encoding.

KEYFRAME_INTERVAL

Description	Specifies the interval for encoding key frames in the bytestream. A key frame has a bytestream information header, a repeated colormap, and uses no interframe escape codes. If KEYFRAME_INTERVAL is set to 0, then no key frames are encoded in the resulting Cell bytestream, and bit-rate control is disabled.
Access	get and set
Type	int
Default	*6*

BITS_PER_SECOND

Description	The bit rate of the resulting Cell bytestream. The rate is guaranteed over a single frame group. If BITS_PER_SECOND is set to 0, then bit rate control is disabled; this is the default. If BITS_PER_SECOND is set to a rate lower than the compressor can produce, then an error is generated, and bit rate control is disabled.
Access	get and set
Type	int *
Default	*0*

COMPRESSOR_MAX_CMAP_SIZE

Description	Sets the maximum colormap size that will be encoded in the Cell bytestream. If COLORMAP_ADAPTION is enabled, this attribute limits the size of the colormaps produced by the compressor. If COLORMAP_ADAPTION is disabled, this attribute limits the size of the colormaps with the COMPRESSOR_COLORMAP attribute. If the compressor is given a colormap that is larger than COMPRESSOR_MAX_CMAP_SIZE , it will be truncated to this length.

The value of this attribute is passed in the Cell bytestream for retrieval with the DECOMPRESSOR_MAX_CMAP_SIZE attribute as an aid to X colormap management.

This attribute can only be set before the first **xil_compress**(3) call. After **xil_compress**(3) has been called or COMPRESSOR_MAX_CMAP_SIZE has been set, it cannot be changed for the life of the XilCis.

Access	get and set
Type	int
Default	*256*

COMPRESSOR_FRAME_RATE

Description	Set the frame rate, in microseconds per frame, at which the images were captured. This value is passed in the Cell bytestream for retrieval with the DECOMPRESSOR_FRAME_RATE attribute. It is permissible to change this attribute in between calls to **xil_compress**(3).
Access	set-only
Type	Xil_unsigned32
Default	*33333 (30 frames/second)*

COMPRESSOR_USER_DATA

Description	Set the user data to be encoded with the next frame. This attribute clears itself after every call to **xil_compress**(3), so it only affects the very next call to **xil_compress** (). A copy of the data is made when setting this attribute, so no assumptions are made about the validity of the data pointer after the attribute is set. The given data is encoded into the Cell bytestream, making the data available to a decompressor via the DECOMPRESSOR_USER_DATA attribute. The attribute accepts a pointer to XilCellUserData, which is a structure containing a pointer to the data and the length of the data. The length of the data is limited to 8K (8192 bytes) per frame. It is permissible to change this attribute in between calls to **xil_compress**(3).
Access	set-only

Type

```
typedef struct {
        Xil_unsigned8*    data;
        Xil_unsigned32    length;
} XilCellUserData;
```

Default Not set

Cell Decompression Attributes

DECOMPRESSOR_COLORMAP

Description	In the case of set, give the Cell decompressor a look-up table with which to perform accelerated 8-bit display of the decompressed image when using **xil_nearest_color**(3). All colormap indices are assumed to be read-only by the decompressor (see RDWR_INDICES). In the case of get, it returns the look-up table associated with the Cell decompressor. This table could possibly have been modified by a call to **xil_decompress**(3). If this attribute has not been set, then it returns NULL.
Access	get and set
Type	XilLookup
Default	*Not set*

RDWR_INDICES

Description	Set the list of colormap indices in the DECOMPRESSOR_COLORMAP look-up table that the Cell decompressor can change for optimum display of decompressed images. The DECOMPRESSOR_MAX_CMAP_SIZE attribute can be used to determine the number of colormap entries needed for optimum display. Setting the list is not cumulative; the list from any previously set attribute call is discarded. Any indices outside the range of the DECOMPRESSOR_COLORMAP look-up table are discarded. Entries in the lookup are only changed on a call to **xil_decompress**(3). If you set this attribute, the Cell decompressor assumes that after each call to **xil_decompress**(3), you will check to see if the XilLookup has been changed via **xil_lookup_get_version**(3), and if so, that you will install the changed colormap before calling **xil_nearest_color**(3) with the XilLookup. Refer to the *XIL Programmer's Guide.* for more details.
Access	set-only
Type	`typedef struct { Xil_unsigned32* pixels; Xil_unsigned16 ncolors; } XilIndexList;`
Default	*Not set*

DECOMPRESSOR_MAX_CMAP_SIZE

Description	Get the maximum size of a colormap for this Cell bytestream. This assists in X colormap management when decompressing the bytestream. Refer to the example in the *XIL Programmer's Guide.* for more information.
Access	get-only
Type	int
Default	*256*

DECOMPRESSOR_FRAME_RATE

Description	Get the frame rate, in microseconds per frame, at which the images were captured. This value is stored in the Cell bytestream via the COMPRESSOR_FRAME_RATE attribute, and is useful only when the compressed image sequence represents a movie. This attribute may have different values at various points in the Cell bytestream if the COMPRESSOR_FRAME_RATE attribute was changed during the creation of the compressed image sequence. If the Cell bytestream does not contain a frame rate, the default value (33333) is returned.
Access	get-only
Type	Xil_unsigned32
Default	*33333 (30 frames/second)*

DECOMPRESSOR_USER_DATA

Description	Get the user data that may be encoded with the most-recently decompressed frame. This attribute clears itself after every call to **xil_decompress**(3), so the returned data is only valid until the next call to **xil_decompress** (). The data decoded from the Cell bytestream was encoded via the COMPRESSOR_USER_DATA attribute. A *pointer* to XilCellUserData is returned.
Access	get-only
Type	XilCellUserData*
Default	Not set

EXAMPLES

The following example opens and closes a Cell CIS using the XIL library:

```
XilSystemState State;
XilCis cis;
State = xil_open();
cis = xil_cis_create(State, "Cell");

-- calls to Cell-specific compression routines --

xil_cis_destroy(cis);
xil_close(State);
```

The following example sets a Cell CIS attribute called *TEMPORAL_FILTERING* to TRUE. Note that because this attribute is not a structure, it is not necessary to pass the address of *TEMPORAL_FILTERING* when setting it.

```
XilCis cis;

xil_cis_set_attribute(cis,"TEMPORAL_FILTERING", (void *) TRUE);
```

The following example returns the value of a Cell CIS attribute called *TEMPORAL_FILTERING.* Note that when getting an attribute it is always necessary to pass the address.

```
Xil_boolean encode_type;
XilCis cis;

xil_cis_get_attribute(cis, "TEMPORAL_FILTERING", (void **) &encode_type);
```

NOTES

The **xil_cis_set_attribute** () and **xil_cis_get_attribute** () calls are used to modify the default behavior of a specific compressor. Generic attributes of compressors are set by individual function calls.

SEE ALSO

xil_cis_create(3), **xil_cis_get_attribute**(3), **xil_cis_get_bits_ptr**(3), **xil_compress**(3), **xil_decompress**(3), **xil_choose_colormap**(3), **xil_lookup_get_version**(3), **xil_nearest_color**(3).

NAME

CellB – XIL driver for CellB video compression / decompression

DESCRIPTION

CellB is a video compression format based on the techniques of block truncation coding and vector quantization. It is well suited for video conferencing. Even though it uses interframe compression, it guarantees that all cells are intraframe encoded periodically, allowing for dropped frames.

Creating a CellB CIS

To compress a compressed image sequence (CIS) with the XIL CellB compressor, specify "CellB" for the *compressorname* argument in **xil_cis_create**(3).

Getting and Setting CellB Attributes

Use **xil_cis_get_attribute**(3) and **xil_cis_set_attribute**(3) to get and set CellB CIS attributes. These attributes are as described in the following sections. Refer to the example section for additional information.

CellB Attributes

The following paragraphs describe the CellB CIS attributes available with the XIL library. All structures and enumerations are defined via **xil.h.** Note that some attributes are "set-only" and others are "get-only." This is noted under the *Access* heading for each attribute.

Note that if you are setting an attribute and that attribute is a structure, you must pass the address of that structure. If you are getting an attribute, you always pass its address.

WIDTH

Description	Sets the frame width of the encoded bitstream. It is only necessary to set this attribute to decompress a bitstream that has been input via a call to **xil_cis_put_bits**(3) or **xil_cis_put_bits_ptr**(3).
Access	set
Type	integer

HEIGHT

Description	Sets the frame height of the encoded bitstream. It is only necessary to set this attribute to decompress a bitstream that has been input via a call to **xil_cis_put_bits**(3) or **xil_cis_put_bits_ptr**(3).
Access	set
Type	integer

IGNORE_HISTORY

Description	CellB bitstreams do not contain any frames containing all the information necessary to reproduce the entire image. In general, this means that these bitstreams are not randomly seekable, because it is expensive to back up far enough so that all cells / macroblocks can be properly decoded for the frame you want to seek to.

By setting *IGNORE_HISTORY* to TRUE, you can build an application that compresses images and decompresses them immediately for display. Setting this attribute to TRUE also allows faster seeking for applications that need to move forward and backward in the bitstream.

Values	*FALSE:* the decoder sets the *RandomAccess* attribute of such CISs to FALSE (i.e., **xil_cis_get_random_access**(3) returns FALSE), and it becomes impossible to seek backwards. Also, seeks forward will actually decode all intermediate frames, instead of just jumping to the appropriate location and decoding the sought frame. *TRUE:* (i.e., **xil_cis_get_random_access**(3) returns TRUE), seeking backwards is possible, and forward seeks may not decode the intermediate frames. After an *IGNORE_HISTORY* seek, the decoded picture may have some bad cells (macroblocks). As these are encoded in subsequent frames, these will "twinkle" in.
Type	Boolean
Access	set / get
Default	FALSE

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

The following example opens and closes a CellB CIS using the XIL library:

```
XilSystemState State;
XilCis cis;
State = xil_open();
cis = xil_cis_create(State, "CellB");

-- calls to CellB-specific compression routines --

xil_cis_destroy(cis);
xil_close(State);
```

NOTE

The CellB bitstream definition (unlike the one for **H261**(3)) does not define a maximum number of frames before a cell must be encoded in the bitstream. However, the encoder that comes with the XIL library does enforce this behavior.

SEE ALSO

xil_cis_get_attribute(3), **xil_cis_create**(3), **xil_cis_put_bits**(3), **xil_cis_put_bits_ptr**(3), **xil_cis_get_bits_ptr**(3), **xil_compress**(3), **xil_decompress**(3).

NAME

faxG3, faxG4 – CCITT Group 3 and Group 4 compressors for compressed image sequences

DESCRIPTION

The XIL library provides compressors that conform to the specifications developed by the Consultative Committee of International Telegraph and Telephone (CCITT) for Group 3 and Group 4 facsimile devices. These standards are supported in the XIL library as defined in recommendations T.4 and T.6 of Fascicle VII.3 (blue book) with the following exceptions: 2-dimensional coding and decoding for Group 3 devices is not currently supported, and no optional extension modes for group 4 coding and decoding are supported. Support for these exceptions may occur in future releases.

These compression techniques, orginially formulated for facsimile devices, are now heavily used by makers of general document storage and retrieval systems. The XIL library's CCITT Group 3 compressor (faxG3) uses a run-length encoding technique; the Group 4 (faxG4) compressor relies almost entirely on a two-dimensional technique. On standard text, the XIL library's Group 3 compressor achieves a compression ratio of about 5:1, and the library's Group 4 compressor achieves a ratio of about 10:1. For more information on these compressors, consult the *XIL Programmer's Guide.*

Creating a CIS

To compress a compressed image sequence (CIS) with an XIL fax compressor, specify either "faxG3" or "faxG4" for the *compressorname* argument in **xil_cis_create**(3).

Getting and Setting Fax Attributes

Although other compression standards include size information (the image width, height, and number of bands) within the data bitstream, the fax standards do not. Thus, if you put compressed data into your CIS using **xil_cis_put_bits**(3) or **xil_cis_put_bits_ptr**(3) you must set the decompressor attributes for width, height, and number of bands; otherwise a call to **xil_decompress**(3) generates an error.

Use **xil_cis_get_attribute**(3) and **xil_cis_set_attribute**(3) to get and set the fax decompression attributes.

Fax Decompression Attributes

The following paragraphs describe the faxG3 and faxG4 CIS attributes available with the XIL library. All structures and enumerations are defined in **xil.h.** These attributes are "set-only," as indicated under the *Access* heading for each attribute.

To set an attribute that is a structure, you must pass that structure's address. To get an attribute, you always pass its address.

WIDTH

Description	defines width of image for fax decompressor
Access	set-only
Type	short
Values	*0 - 32767*
Default	*0*
Notes	Set the value of this attribute to the width in pixels of the images to be decompressed. If you do not set it, its value is 0 and an error occurs when you call **xil_decompress**(3), as discussed above in "Getting and Setting Fax Attributes."

HEIGHT

Description	defines height of image for fax decompressor
Access	set-only
Type	short
Values	*0 - 32767*
Default	*0*
Notes	Set the value of this attribute to the height in pixels of the images to be decompressed. If you do not set it, its value is 0 and an error occurs when you call **xil_decompress**(3), as discussed above in "Getting and Setting Fax Attributes."

BANDS

Description	defines number of bands in image for fax decompressor
Access	set-only
Type	short
Values	*0 - 32767*
Default	*0*
Notes	Set the value of this attribute to the number of bands in the images to be decompressed. If you do not set it, its value is 0 and an error occurs when you call **xil_decompress**(3), as discussed above in "Getting and Setting Fax Attributes."

EXAMPLES

The following example opens and closes a faxG3 CIS using the XIL library:

```
XilSystemState State;
XilCis cis;
State = xil_open();
cis = xil_cis_create(State, "faxG3");

-- calls to faxG3-specific compression routines --

xil_cis_destroy(cis);
xil_close(State);
```

NOTES

The **xil_cis_set_attribute** () and **xil_cis_get_attribute** () calls are used to modify the default behavior of a specific compressor. Generic attributes of compressors are set by individual function calls.

SEE ALSO

xil_cis_create(3), **xil_cis_get_attribute**(3), **xil_cis_get_bits_ptr**(3), **xil_compress**(3), **xil_decompress**(3), **xil_cis_put_bits**(3), **xil_cis_put_bits_ptr**(3).

NAME

H261 – H.261 decompressor for compressed image sequences

DESCRIPTION

CCITT Recommendation H.261, Video Codec for Audiovisual Services at p x 64 kbit/s, is an international standard for videophone and videoconferencing. It describes the moving picture component of audiovisual services at the rates of p x 64 kbit/s, where p is in the range 1 to 30.

The XIL H261 codec implements the H.261 standard without the transmission coder/decoder; i.e., the XIL bitstream does not contain any Error Correction Framing bits.

The current release of the XIL library does not contain an implementation of an H.261 compressor. Calls to **xil_compress**(3) will produce an error unless a third party H.261 compressor has been installed.

Image Types

The H261 decompressor produces 3-band, XIL_BYTE images in the XIL library's "ycc601" color space (The XIL image color space will not be examined or set by the H261 codec, but the codec assumes its input image has the proper color space). The width and height of the images must be either Common Intermediate Format (CIF), which is 352 wide by 288 high, or Quarter CIF (QCIF), which is 176 wide by 144 high.

Creating a CIS

To create an H.261 compressed image sequence (CIS), specify "H261" for the *compressorname* argument in **xil_cis_create**(3).

Getting and Setting H261 Attributes

Use **xil_cis_get_attribute**(3) and **xil_cis_set_attribute**(3) to get and set H261 CIS attributes. These attributes are as described in the following sections. Refer to the example section for additional information.

H261 Compression Attributes

The following paragraphs describe the H.261 CIS attributes available with the XIL library. All structures and enumerations are defined via **xil.h.** Note that if you are setting an attribute and that attribute is a structure, you must pass the address of that structure. If you are getting an attribute, you always pass the address of the attribute. If you are getting a structure attribute, you must pass a pointer to a pointer to the structure and XIL will set the pointer to the structure. You must free the memory for this structure (using free(3C)) when it is no longer needed.

COMPRESSOR_BITS_PER_IMAGE

Description	Encode images with this number of bits per image. This is normally bits_per_second/frames_per_second.
Access	get and set
Type	Integer
Values	value must be greater than or equal to 0
Default	5069 (0.2 bits/pixel at QCIF resolution)

COMPRESSOR_IMAGE_SKIP

Description	Number of images skipped between compressed images. Controls the Temporal Reference counter in the bitstream. (Temporal Reference is incremented by 1 + COMPRESSOR_IMAGE_SKIP)
Access	get and set
Type	Integer
Values	0-31
Default	0

COMPRESSOR_MV_SEARCH_RANGE

Description: Set motion vector search range. Value 15 is the maximum H.261 search range. Value 0 means that the search range is limited to the spatially corresponding block in the previous picture. This attribute is only a suggestion and may be ignored by the compressor. It may be used to speed up compression at the expense of compression quality.

Access: get and set

Type:

```
typedef struct {
        int x;    /* horizontal search limit */
        int y;    /* vertical search limit */
} XilH261MVSearchRange;
```

Values: *x:* Can have a value in the range of 0-15

y: Can have a value in the range of 0-15

Default: 15 for both x and y

COMPRESSOR_LOOP_FILTER

Description	Allow encoder to use loop filtering. This attribute is only a suggestion and may be ignored by the compressor. It may be used to speed up compression (and / or decompression of the resulting bitstream) at the expense of compression quality.
Access	get and set
Type	Xil_boolean
Values	*TRUE:* Loop filtering turned on *FALSE:* Loop filtering turned off
Default	*TRUE*

COMPRESSOR_ENCODE_INTRA

Description	Cause encoder to encode pictures in INTRA mode with coding parameters to avoid buffer overflow. (This attribute can be used by the application in response to a Fast Update signal sent via H.221).
Access	get and set
Type	Xil_boolean
Values	*TRUE:* Intra-only coding turned on *FALSE:* Intra-only coding turned off
Default	*FALSE*

COMPRESSOR_FREEZE_RELEASE

Description	Set the Freeze Picture Release bit in each picture in the bitstream, starting with the next compressed picture.
Access	get and set
Type	Xil_boolean
Values	*TRUE:* Set the Freeze Picture Release bit in the bitstream. *FALSE:* Do not set the Freeze Picture Release bit in the bitstream.
Default	*FALSE*

COMPRESSOR_SPLIT_SCREEN

Description	Set the Split Screen Indicator bit in each picture in the bitstream, starting with the next compressed picture.
Access	get and set
Type	Xil_boolean
Values	*TRUE:* Set the Split Screen Indicator bit in the bitstream. *FALSE:* Do not set the Split Screen Indicator bit in the bitstream.
Default	*FALSE*

COMPRESSOR_DOC_CAMERA

Description	Set the Document Camera Indicator bit in each picture in the bitstream, starting with the next compressed picture.
Access	get and set
Type	Xil_boolean
Values	*TRUE:* Set the Document Camera Indicator bit in the bitstream. *FALSE:* Do not set the Document Camera Indicator bit in the bitstream.

Default	*FALSE*

H261 Decompression Attributes

IGNORE_HISTORY

Description	If TRUE, perform forward seeks without updating the decoding history and allow backward seeking (decompression after these seeks may yield incomplete results). If FALSE, maintain proper decoding history during forward seeks and disallow backward seeking. **xil_cis_get_random_access**(3) will return TRUE if *IGNORE_HISTORY* is TRUE, and will return FALSE if *IGNORE_HISTORY* is FALSE.
Access	get and set
Type	Xil_boolean
Values	*TRUE:* Allow backward seeks and perform fast forward seeks. *FALSE:* Perform correct seeking.
Default	*FALSE*

DECOMPRESSOR_FREEZE_RELEASE

Description	Return value of the Freeze Picture Release bit from the picture header of the most recently decompressed picture. Value is available immediately after executing an **xil_decompress**(3) call and may be "gotten" and tested without compromising the execution of a decompression molecule.
Access	get
Type	Xil_boolean
Values	*TRUE:* Freeze Picture Release bit is set. *FALSE:* Freeze Picture Release bit is not set.
Default	Value is undefined if no pictures have been decompressed.

DECOMPRESSOR_SPLIT_SCREEN

Description	Return value of the Split Screen Indicator bit from the picture header of the most recently decompressed picture. Value is available immediately after executing an **xil_decompress**(3) call and may be "gotten" and tested without compromising the execution of a decompression molecule.
Access	get
Type	Xil_boolean
Values	*TRUE:* Split Screen Indicator bit is set. *FALSE:* Split Screen Indicator bit is not set.

Default	Value is undefined if no pictures have been decompressed.

DECOMPRESSOR_DOC_CAMERA

Description	Return value of the Document Camera Indicator bit from the picture header of the most recently decompressed picture. Value is available immediately after executing an **xil_decompress**(3) call and may be "gotten" and tested without compromising the execution of a decompression molecule.
Access	get
Type	Xil_boolean
Values	*TRUE:* Document Camera Indicator bit is set. *FALSE:* Document Camera Indicator bit is not set.
Default	Value is undefined if no pictures have been decompressed.

DECOMPRESSOR_SOURCE_FORMAT

Description	Return value of the Source Format bit from the picture header of the most recently decompressed picture. Value is available immediately after executing an **xil_decompress**(3) call and may be "gotten" and tested without compromising the execution of a decompression molecule.
Access	get

Type	typedef enum {
	QCIF, CIF
	} XilH261SourceFormat;

Values	*CIF:* Source Format (picture size) is Common Intermediate Format (CIF) *QCIF:* Source Format (picture size) is Quarter Common Intermediate Format (QCIF)
Default	Value is undefined if no pictures have been decompressed.

DECOMPRESSOR_TEMPORAL_REFERENCE

Description	Return value of the Temporal Reference from the picture header of the most recently decompressed picture. Temporal Reference is formed by incrementing its value in the previously transmitted picture header by one plus the number of non-transmitted pictures (at 29.97 Hz) since the last transmitted one. Arithmetic is performed modulo 32. Value is available immediately after executing an **xil_decompress**(3) call and may be "gotten" and tested without compromising the execution of a decompression molecule.
Access	get

Type	Integer
Values	*value* can be an integer from 0 to 31.
Default	Value is undefined if no pictures have been decompressed.

EXAMPLES

The following example opens and closes an H.261 CIS using the XIL library:

```
XilSystemState State;
XilCis cis;
State = xil_open();
cis = xil_cis_create(State, "H261");

-- calls to H261-specific compression routines --

xil_cis_destroy(cis);
xil_close(State);
```

The following example sets an H.261 CIS attribute called *COMPRESSOR_LOOP_FILTER* to TRUE. Note that because this attribute is not a structure, it is not necessary to pass the address of *COMPRESSOR_LOOP_FILTER* when setting it.

```
XilCis cis;

xil_cis_set_attribute(cis,"COMPRESSOR_LOOP_FILTER", (void *) TRUE);
```

The following example returns the value of an H.261 CIS attribute called *DECOMPRESSOR_DOC_CAMERA*. Note that when getting an attribute it is always necessary to pass the address.

```
Xil_boolean on;
XilCis cis;

xil_cis_get_attribute(cis, "DECOMPRESSOR_DOC_CAMERA", (void **) &on);
```

NOTES

The **xil_cis_set_attribute** () and **xil_cis_get_attribute** () calls are used to modify the default behavior of a specific compressor. Generic attributes of compressors are set by individual function calls.

SEE ALSO

xil_cis_create(3), **xil_cis_get_attribute**(3), **xil_cis_get_bits_ptr**(3), **xil_compress**(3), **xil_decompress**(3).

NAME

Jpeg – JPEG compressor / decompressor for compressed image sequences

DESCRIPTION

The Joint Photographic Experts Group (JPEG) is a joint ISO / CCITT technical committee. JPEG has developed a general-purpose international standard for the compression of continuous tone (grayscale or color) still images. The standard has three categories:

The baseline specification,
An extended features specification, and
A lossless compression specification.

The baseline specification is a simple coding method based on the Discrete Cosine Transform (DCT) and uniform quantization in combination with statistical Huffman coding techniques for 8-bit image components. The extended features specification provides support for higher image precision, alternate statistical coding techniques, and progressive operation. The lossless compression technique is a two-dimensional predictive differential coding technique, which also combines with statistical coding.

The XIL implementation supports the baseline specification, "Jpeg", and the lossless specification, "JpegLL". Consult **JpegLL**(3) for more information on the XIL library's JPEG Lossless implementation.

Certain combinations of XIL operations are accelerated. These combinations should be used for the highest performance in JPEG decompression. For more information and example programs, see the *XIL Programmer's Guide.*

Creating a CIS

To compress a compressed image sequence (CIS) with the XIL JPEG compressor, specify "Jpeg" for the *compressorname* argument in **xil_cis_create**(3).

Getting and Setting JPEG Attributes

Use **xil_cis_get_attribute**(3) and **xil_cis_set_attribute**(3) to get and set JPEG CIS attributes. These attributes are as described in the following sections. Refer to the example section for additional information.

JPEG Compression Attributes

The following paragraphs describe the JPEG CIS attributes available with the XIL library. All structures and enumerations are defined via **xil.h.** Note that some attributes are "set-only" and others are "get-only." This is noted under the *Access* heading for each attribute.

Note that if you are setting an attribute and that attribute is a structure, you must pass the address of that structure. If you are getting an attribute, you always pass its address.

BAND_HUFFMAN_TABLE

Description	Associates a band of an image to a Huffman table
Access	set-only

Type	typedef struct {
	int band;
	int table;
	XilJpegHTableType type;
	} XilJpegBandHTable;

Values	*band:* Can have a value in the range 0-255. *table:* For Baseline JPEG, you can have a value in the range 0-1. *type:* For Baseline JPEG, you can have a value *DC* or *AC.*
Default	*band 0's DC* component is associated to *table 0, type DC* and *band 0's AC* component is associated to *table 0, type AC.* All other bands' *DC* component is associated to *table 1, type DC,* and their *AC* component is associated to *table 1, type AC.*
Notes	Bands may be associated to tables that have not yet been set. However, the tables must be set before a call to compress is made or an error occurs.

BAND_QUANTIZER

Description	Associates a band of an image to a quantization table
Access	set-only

Type	typedef struct {
	    int band;
	    int table;
	} XilJpegBandQTable;

Values	*band:* Can have a value in the range 0-255. *table:* Can have a value in the range 0-3.
Default	*band 0* is associated to *table 0.* All other bands are associated to *table 1.*
Notes	Bands may be associated to tables that have not yet been set. However, the tables must be set before a call to compress is made or an error occurs.

BYTES_PER_FRAME

Description	Number of bytes in the last frame compressed by a CIS. This value can be used to assist in selecting a *COMPRESSION_QUALITY* to achieve a desired bit rate.
Access	get-only
Type	int
Default	Not applicable. Value is undefined for a CIS that has not compressed any frames.

COMPRESSED_DATA_FORMAT

Description	defines output format for JPEG compressor
Access	set

Type	typedef enum{
	INTERCHANGE, ABBREVIATED_FORMAT
	} XilJpegCompressedDataFormat;

Values	*INTERCHANGE:* Use JPEG interchange format. All quantization and entropy-coding table specifications needed by the decoding process are included in each compressed frame. *ABBREVIATED_FORMAT:* Use JPEG abbreviated format for compressed images. Quantization and entropy-coding table specifications are not included in a compressed frame if the specifications are defined in a previous frame in the compressed sequence. If any table values change after they are defined in the compressed sequence, a new table definition is included in the first compressed frame that uses the new table values.
Default	*ABBREVIATED_FORMAT*
Notes	This does not include the third type: *ABBREVIATED_TABLE,* in which a frame contains only table specifications. However, the decoder will accept this format.

COMPRESSION_QUALITY

Description	Provide a hint to the compressor, enabling it to increase the compression ratio by reducing the compressed image quality.
Access	set-only
Type	int value
Values	*value* can be an integer from 1 to 100. Setting *value* to 100 requests the highest quality achievable by the compressor. A *value* equal to 1 sets the compression ratio to the maximum achievable while reducing quality. The compression ratio may also be affected by modifying the quantization table. However, this attribute will not cause a change in values for the *QUANTIZATION_TABLE.*
Default	50

ENCODE_INTERLEAVED

Description	If the image to compress is composed of 4 bands or less, having this attribute set to TRUE will generate an interleaved JPEG-compliant bitstream (for example, 1:1:1:1 for a 4 banded image). If the number of bands exceeds 4 or if this attribute is set to FALSE, a noninterleaved JPEG-compliant bitstream is generated. Note, that the *ENCODE_411_INTERLEAVED* attribute takes precedence over *ENCODE_INTERLEAVED* for Baseline JPEG.

Access	set-only
Type	Xil_boolean
Values	*TRUE:* For images of 4 bands or less, produce an interleaved JPEG-compliant bitstream. *FALSE:* Produce a noninterleaved JPEG-compliant bitstream.
Default	*TRUE*

ENCODE_411_INTERLEAVED

Description	For Baseline JPEG, if the image to compress is a 3-banded image, setting this attribute to TRUE generates a 2x2:1:1 macroblock JPEG-compliant bitstream. This is useful to gain additional compression for YCbCr images and is mandatory for most decompression molecules. It is not appropriate for RGB images. If an image is not 3-banded, then the *ENCODE_411_INTERLEAVED* attribute is treated as if it were false, and therefore the *ENCODE_INTERLEAVED* attribute controls the interleaved format of the bitstream. Otherwise, the *ENCODE_411_INTERLEAVED* attribute takes precedence over *ENCODE_INTERLEAVED*. Because some decompressor molecules require the bitstream image size to be a multiple of 16 in both width and height, source images should be clipped (for example, by using a child image) before compression, if the highest decompression speed is desired.
Access	set-only
Type	Xil_boolean
Values	*TRUE:* Generate a 2x2:1:1 macroblock, JPEG-compliant bitstream if the input image is a 3-banded image. *FALSE:* Do not generate a 2x2:1:1 macroblock, JPEG-compliant bitstream.
Default	*FALSE*

HUFFMAN_TABLE

Description	Set values in specified Huffman table
Access	set-only
Type	

```
typedef struct {
        int table;
        XilJpegHTableType type;
        XilJpegHTableValue *value;
} XilJpegHTable;
```

```
typedef enum {
        DC, AC
} XilJpegHTableType;

typedef struct {
         int bits;
         int pattern;
} XilJpegHTableValue;
```

Values	*table:* For Baseline JPEG, you can have a value in the range from 0-1. *type:* For Baseline JPEG, you can have a value *DC* or *AC*. It also specifies how many entries are in the value array: 16 if type is *DC;* 256 if type is *AC*. *value:* A pointer to an array of dyads, where the first element 'bits' indicates the number of valid bits in the second element 'pattern'.
Default	By default, the values in each of the tables are pre-initialized to the example values given in Annex K of the ANSI JPEG specification. Default values for *table 0, type DC* are given in Table K.3 and are useful for the *DC* coefficients of the luminance band of 8-bit Y,Cb,Cr images. Default values for *table 1, type DC* are given in Table K.4 and are useful for the *DC* coefficients of the chrominance bands of 8-bit Y,Cb,Cr images. Default values for *table 0, type AC* are given in Table K.5 and are useful for the *AC* coefficients of the luminance band of 8-bit Y,Cb,Cr images. Default values for *table 1, type AC* are given in Table K.6 and are useful for the *AC* coefficients of the chrominance bands of 8-bit Y,Cb,Cr images.

OPTIMIZE_HUFFMAN_TABLES

Description	Provide a hint to the compressor, enabling it to generate optimal Huffman tables instead of using the default example values specified in the ANSI specification. This is only a hint; the compressor is free to ignore the hint.
Access	set-only
Type	Xil_boolean
Values	*TRUE:* Huffman tables may vary from image to image to achieve higher compression.

	FALSE: Use fixed Huffman tables for each image in the sequence.
Default	*FALSE*

QUANTIZATION_TABLE

Description	Set the values in a specific quantization table
Access	set-only
Type	typedef struct { int table; int (∗value)[8]; } XilJpegQTable;
Values	*table:* Can have a value in the range 0-3. *value:* For Baseline JPEG, the 64 quantization table elements are defined to be 8-bit values; the compressor uses the least significant 8 bits of the input table value. This precision assumption may vary according to the compressor / decompressor configuration. The quantization operation for a DCT coefficient uses the corresponding element from the input quantization table.
Default	Default values for *table 0* are given in Table K.1 of Annex K of the ANSI JPEG specification, and are useful for the luminance band of 8-bit Y,Cb,Cr images. The default values for *table 1* are given in Table K.2 of Annex K of the ANSI JPEG specification, and are useful for the chrominance bands of 8 bit Y,Cb,Cr images. *tables 2* and *table 3* are not loaded with any values.
Notes	A table that is to be used to compress an image must be set before the call to compress. The compressor issues an error if a band has been set to use a particular quantization table that has not yet been set.

TEMPORAL_FILTERING

Description	Turns on or off a form of temporal filtering that may reduce noise in video sequences. The filtering may also introduce undesirable artifacts in sequences containing motion. Filtering is only performed on 3-banded images.
Access	get and set
Type	Xil_boolean
Values	*TRUE:* Filtering turned on *FALSE:* Filtering turned off
Default	FALSE

JPEG Decompression Attributes

DECOMPRESSION_QUALITY

Description	Provide a suggestion to the decompressor, enabling it to trade off reconstruction quality in exchange for an increase in decoding speed.
Access	set
Type	int value
Values	*value* can be between 1 and 100. A *value* of 100 sets the quality to maximum. A *value* of 1 sets the speed to its maximum and allows the quality to decrease to the minimum allowed by the decompressor. The decompressor is free to ignore this hint.
Default	100
Notes	The JPEG decompressor will increase speed by decreasing the number of quantized coefficients that it uses in reconstruction.

IGNORE_HISTORY

Description	Some JPEG bitstreams contain images that define tables (Huffman and/or Quantization) and images that use tables defined in previous images. These bitstreams are not, in general, randomly seekable, because it is possible to backup to a point where the required tables for decoding the next image have not been loaded into the decoder. The JPEG decoder detects such bitstreams.
Access	set-only
Type	Xil_boolean
Values	*FALSE:* The decoder will set the *RandomAccess* attribute of such CISs to FALSE (i.e., **xil_cis_get_random_access**(3) returns FALSE), and it is impossible to seek backwards. *TRUE:* The function **xil_cis_get_random_access**(3) returns TRUE, regardless of the type of bitstream, and it is always possible to seek backwards. If *IGNORE_HISTORY* is set to TRUE, the application should not seek forward beyond frames that contain table definitions if those definitions are needed for subsequent decoding; the decoder will not ensure that these table definitions are loaded.
Default	*FALSE*
Notes	If this attribute is set to TRUE, it is the responsibility of the application to seek to a spot in the bitstream that will decode correctly (either the image defines its own tables, or it depends on tables that have been most recently loaded into the decoder). If you have a CIS that is randomly seekable and you never back up, you can seek forward to any frame and get the correct answer, regardless of the setting of *IGNORE_HISTORY*.

If you have a CIS that is randomly seekable and if *IGNORE_HISTORY* is FALSE, you can seek forward to any frame and get the correct answer (regardless of whether you have ever backed up).

If you have a CIS that is randomly seekable and if the attribute is TRUE, and if you have ever backed up, you need to control the loading of Q tables yourself by explicitly decompressing the frames that contain the tables. This is only practical if (1) you are decompressing every frame, or (2) you know the location of the Q tables because you glued together two CISs so that you know the location of the boundary.

EXAMPLES

The following example opens and closes a JPEG CIS using the XIL library:

```
XilSystemState State;
XilCis cis;
State = xil_open();
cis = xil_cis_create(State, "Jpeg");

-- calls to Jpeg-specific compression routines --

xil_cis_destroy(cis);
xil_close(State);
```

The following example sets a JPEG CIS attribute called *ENCODE_INTERLEAVED* to TRUE. Note that because this attribute is not a structure, it is not necessary to pass the address of *ENCODE_INTERLEAVED* when setting it.

```
XilCis cis;

xil_cis_set_attribute(cis,"ENCODE_INTERLEAVED", (void *) TRUE);
```

The following example returns the value of a JPEG CIS attribute called *ENCODE_INTERLEAVED*. Note that when getting an attribute it is always necessary to pass the address.

```
Xil_boolean encode_type;
XilCis cis;

xil_cis_get_attribute(cis, "ENCODE_INTERLEAVED", (void **) &encode_type);
```

NOTES

The **xil_cis_set_attribute**() and **xil_cis_get_attribute**() calls are used to modify the default behavior of a specific compressor. Generic attributes of compressors are set by individual function calls.

SEE ALSO

xil_cis_create(3), **xil_cis_get_attribute**(3), **xil_cis_get_bits_ptr**(3), **xil_compress**(3), **xil_decompress**(3).

NAME

JPEG Lossless compressor / decompressor – JPEG Lossless compressor / decompressor for compressed image sequences

DESCRIPTION

The Joint Photographic Experts Group (JPEG) is a joint ISO / CCITT technical committee. JPEG has developed a general-purpose international standard for the compression of continuous tone (grayscale or color) images. The standard has three categories:

The baseline specification,
An extended features specification, and
A lossless compression specification.

The baseline specification is a simple coding method based on the Discrete Cosine Transform (DCT) and uniform quantization in combination with statistical Huffman coding techniques for 8-bit image components. The extended features specification provides support for higher image precision, alternate statistical coding techniques, and progressive operation. The lossless compression technique is a 2-dimensional predictive differential coding technique, which also combines with statistical coding.

For more information and example programs, see the *XIL Programmer's Guide.*

The XIL implementation supports the baseline specification, "Jpeg" and the lossless specification, "JpegLL". Consult **Jpeg**(3) for more information on the XIL library's JPEG implementation.

Creating a CIS

To compress a compressed image sequence (CIS) with the XIL JPEG Lossless compressor, specify "JpegLL" for the *compressorname* argument in **xil_cis_create**(3).

Getting and Setting Attributes

Use **xil_cis_get_attribute**(3) and **xil_cis_set_attribute**(3) to get and set JPEG Lossless CIS attributes. These attributes are described in the following sections. Refer to the example section for additional information.

JpegLL Compression Attributes

The following paragraphs describe the JPEG Lossless CIS attributes available with the XIL library. All structures and enumerations are defined via **xil.h.** Note that some attributes are "set-only" and others are "get-only." This is noted under the *Access* heading for each attribute.

Note that if you are setting an attribute and that attribute is a structure, you must pass the address of that structure. If you are getting an attribute, you always pass its address.

COMPRESSED_DATA_FORMAT

Description	defines output format for JPEG Lossless compressor
Access	set
Type	typedef enum{ INTERCHANGE, ABBREVIATED_FORMAT } XilJpegCompressedDataFormat;

Values	*INTERCHANGE:* Use JPEG interchange format. All quantization and entropy-coding table specifications needed by the decoding process are included in each compressed frame. *ABBREVIATED_IMAGE:* Use JPEG abbreviated format for compressed images. Quantization and entropy-coding table specifications are not included in a compressed frame if the specifications are defined in a previous frame in the compressed sequence. If any table values change after they are defined in the compressed sequence, a new table definition is included in the first compressed frame that uses the new table values.
Default	*ABBREVIATED_IMAGE*
Notes	This does not include the third type: *ABBREVIATED_TABLE*, in which a frame contains only table specifications. However, the decoder will accept this format.

ENCODE_INTERLEAVED

Description	If the image to compress is composed of 4 bands or less, having this attribute set to TRUE will generate an interleaved JPEG-compliant bitstream (for example, 1:1:1:1 for a 4 banded image). If the number of bands exceeds 4 or if this attribute is set to FALSE, a noninterleaved JPEG-compliant bitstream is generated.
Access	set-only
Type	Xil_boolean
Values	*TRUE:* For images of 4 bands or less, produce an interleaved JPEG-compliant bitstream. *FALSE:* Produce a noninterleaved JPEG-compliant bitstream.
Default	*TRUE*

HUFFMAN_TABLE

Description	Set values in specified Huffman table
Access	set-only
Type	see below

```
typedef struct {
        int table;
        XilJpegHTableType type;
        XilJpegHTableValue *value;
} XilJpegHTable;

typedef enum {
        DC, AC
} XilJpegHTableType;
```

```
typedef struct {
        int bits;
        int pattern;
} XilJpegHTableValue;
```

Values	*table:* For Lossless JPEG, you can have a value in the range from 0-3. *type:* For Lossless JPEG, you must have value *DC*, and there are 17 entries in the value array. *value:* A pointer to an array of dyads, where the first element ‘bits’ indicates the number of valid bits in the second element ‘pattern’.
Default	By default, the values in each of the tables are pre-initialized to the example values given in Annex K of the ANSI JPEG specification. Default values for *table 0, type DC* are given in Table K.3 and are useful for the *DC* coefficients of the luminance band of 8-bit Y,Cb,Cr images. Default values for *table 1, type DC* are given in Table K.4 and are useful for the *DC* coefficients of the chrominance bands of 8-bit Y,Cb,Cr images. Default values for *table 0, type AC* are given in Table K.5 and are useful for the *AC* coefficients of the luminance band of 8-bit Y,Cb,Cr images. Default values for *table 1, type AC* are given in Table K.6 and are useful for the *AC* coefficients of the chrominance bands of 8-bit Y,Cb,Cr images.

BAND_HUFFMAN_TABLE

Description	Associates a band of an image to a Huffman table
Access	set-only
Type	`typedef struct { int band; int table; XilJpegHTableType type; } XilJpegBandHTable;`
Values	*band:* Can have a value in the range 0-255. *table:* For Lossless JPEG, you can have a value in the range 0-3. *type:* For Lossless JPEG, you must have value *DC*.

Default	*band 0's DC* component is associated to *table 0, type DC* and *band 0's AC* component is associated to *table 0, type AC.* All other bands' *DC* component is associated to *table 1, type DC,* and their *AC* component is associated to *table 1, type AC.*
Notes	Bands may be associated to tables that have not yet been set. However, the tables must be set before a call to compress is made or an error occurs.

OPTIMIZE_HUFFMAN_TABLES

Description	Provide a hint to the compressor, enabling it to generate optimal Huffman tables instead of using the default example values specified in the ANSI specification. This is only a hint; the compressor is free to ignore the hint. For Lossless JPEG, setting this option attribute on or off keeps the current tables loaded. No optimal Huffman tables are provided for Lossless JPEG.
Access	set-only
Type	Xil_boolean
Values	*TRUE:* Huffman tables may vary from image to image to achieve higher compression. *FALSE:* Use fixed Huffman tables for each image in the sequence.
Default	*FALSE*

LOSSLESS_BAND_SELECTOR

Description: Associates a band of an image to a predictor selection for the Lossless JPEG compressor. In the following discussion under the *Values* heading, Px = prediction for pixel "x", A = pixel left, B = pixel above, C = pixel diagonally above and left.

```
C  B
A  x
```

Access: set-only

Type:

```
typedef struct {
            int band;
            XilJpegLLBandSelectorType selector;
} XilJpegLLBandSelector;

typedef enum {
                  ONE_D1,ONE_D2,ONE_D3,TWO_D1,TWO_D2,
                           TWO_D3,TWO_D4
 }  XilJpegLLBandSelectorType;
```

Values	*band:* Can have a value in the range 0-255.
	NO_PRED: Invalid selection for Lossless JPEG.
	ONE_D1: Px = A
	ONE_D2: Px = B
	ONE_D3: Px = C
	TWO_D1: Px = A + B - C
	TWO_D2: Px = A + ((B - C) / 2)
	TWO_D3: Px = B + ((A - C) / 2)
	TWO_D4: Px = (A + B) / 2
Default	All bands default to selector ONE_D1.

LOSSLESS_BAND_PT_TRANSFORM

Description	Associates a band of an image with a point transform, PtTransform, for the Lossless JPEG compressor. If PtTransform is non-zero, the input image band is divided by 2**PtTransform before lossless encoding.
Access	set-only
Type	typedef struct { int band; int PtTransform; } XilJpegLLBandPtTransform;
Values	*band:* Can have a value in the range 0-255. *PtTransform:* Can have a value in the range 0-15.
Default	All bands default to PtTransform = 0.

EXAMPLES

The following example opens and closes a JPEG Lossless CIS using the XIL library:

```
XilSystemState State;
XilCis cis;
State = xil_open();
cis = xil_cis_create(State, "JpegLL");

-- calls to JpegLL-specific compression routines --

xil_cis_destroy(cis);
xil_close(State);
```

The following example sets a JPEG Lossless CIS attribute called *ENCODE_INTERLEAVED* to TRUE. Note that because this attribute is not a structure, it is not necessary to pass the address of *ENCODE_INTERLEAVED* when setting it.

```
XilCis cis;

xil_cis_set_attribute(cis,"ENCODE_INTERLEAVED", (void *) TRUE);
```

The following example returns the value of a JPEG Lossless CIS attribute called *ENCODE_INTERLEAVED.* Note that when getting an attribute it is always necessary to pass the address.

```
Xil_boolean encode_type;
XilCis cis;

xil_cis_get_attribute(cis, "ENCODE_INTERLEAVED", (void **) &encode_type);
```

NOTES

The **xil_cis_set_attribute** () and **xil_cis_get_attribute** () calls are used to modify the default behavior of a specific compressor. Generic attributes of compressors are set by individual function calls.

SEE ALSO

xil_cis_create(3), **xil_cis_get_attribute**(3), **xil_cis_get_bits_ptr**(3), **xil_compress**(3), **xil_decompress**(3).

NAME

Mpeg1 – MPEG decompressor for compressed image sequences

DESCRIPTION

The Moving Picture Experts Group (MPEG), an ISO technical committee, has developed a general-purpose international standard for the compression of full motion video to a bit rate of 1.5 Mbits/second. The method employs transform coding, specifically the Discrete Cosine Transform (DCT), to obtain intraframe compression by reducing spatial redundancy, and motion compensation to obtain interframe compression by reducing temporal redundancy.

The XIL implementation supports the basic MPEG specification for video compression, but does not address audio and synchronization issues. Certain combinations of XIL operations are accelerated. These combinations should be used for the highest performance in MPEG decompression. For more information and example programs, see the *XIL Programmer's Guide.*

For a bitstream with B frames, the behavior of the **xil_cis_get_bits_ptr**(3) function differs from its usual behavior. For more information, see the discussion of B pictures in the *XIL Programmer's Guide.*

The current release of the XIL library does not contain an implementation of an MPEG compressor. Calls to **xil_compress**(3) will produce an error unless a third-party MPEG compressor has been installed. Also, streams with D frames will not be decompressed.

Creating a CIS

To decompress a compressed image sequence (CIS) with the XIL MPEG decompressor, specify "Mpeg1" for the *compressorname* argument in **xil_cis_create**(3).

Getting and Setting MPEG Attributes

Use **xil_cis_get_attribute**(3) and **xil_cis_set_attribute**(3) to get and set MPEG CIS attributes. These attributes are as described in the following sections. Refer to the example section for additional information.

MPEG Compression Attributes

The following paragraphs describe the MPEG CIS attributes available with the XIL library. All structures and enumerations are defined in **xil.h.** Note that all compression attributes are "settable" and "gettable."

Note that if you are setting an attribute and that attribute is a structure, you must pass the address of that structure. If you are getting an attribute, you always pass the address of the attribute. If you are getting a structure attribute, you must pass a pointer to a pointer to the structure, and XIL will set the pointer to the structure. You must free the memory for this structure (using **free** (3C)) when it is no longer needed.

Many of these attributes employ a "null default" (ND) convention under which setting an attribute to zero/null signifies that the compressor is allowed (required) to use a value that is optimal for its purposes. In all cases, the zero/null value would not otherwise be legal. ND attributes are "gettable" in the sense that they will return null/zero if they are so set, but are opaque with regard to the actual default value used by the compressor. In addition, all ND attributes have null/zero as the default.

COMPRESSOR_BITS_PER_SECOND

Description	Controls the output data rate of the MPEG bitstream in bits/second.
Access	set/get
Type	int value
Values	1 - 104,856,800, rounded upward to the nearest multiple of 400.
Default	1,152,000
Notes	Cannot be changed after the first frame is compressed. Should be set to no more than 1,856,000 if a Constrained Parameter bitstream is desired.

COMPRESSOR_INSERT_VIDEO_SEQUENCE_END

Description	Causes a Video Sequence End code to be inserted into the bitstream.
Access	set/get
Type	Xil_boolean
Values	FALSE - no end code inserted. TRUE - end code inserted after each subsequent call to **xil_cis_flush**(3), assuming this attribute value remains TRUE. Inserting the code is done in addition to the normal actions of the flush routine. When set to FALSE, this attribute doesn't affect the normal actions of the flush routine. The library prevents multiple end codes from being written to the same frame.
Default	FALSE
Notes	An MPEG-1 sequence isn't valid without the end code; therefore, the last frame in the sequence must be followed by the code. Since it cannot predict when an application will end a sequence, the MPEG-1 codec reserves the last frame or subgroup of frames in the CIS so the application can write the end code to that reserved frame. The frame or subgroup must be released before it can be retrieved with **xil_cis_get_bits_ptr**(3) or **xil_decompress**(3). This affects the logic used when making these calls, and also affects the logic used with loop continuation conditions that call **xil_cis_has_frame**(3) to control CIS decompression. A frame or subgroup is released under either of two conditions: when it's followed by an end code, or when it's followed by another frame or subgroup. See the *Xil Programmer's Guide* for more information on releasing reserved frames. There can be multiple video-sequence headers associated with one end code, since the header information changes as certain CIS attributes change (within the XIL limitations that there are no width/height changes). In addition, there may be multiple sequences within a bitstream. If frames are compressed into the CIS after the call to **xil_cis_flush** (3),

it's the compressor's responsibility to provide the video-sequence header information per sequence. Before the application changes an attribute that would result in a new sequence header, it must first output an end code for the current sequence.

COMPRESSOR_INTRA_QUANTIZATION_TABLE

Description	Set quantization matrix for I-frame compression.
Access	set / get
Type	Xil_unsigned8[64]
Values	1 - 255
Default	ND = null
Notes	Set by passing a pointer to an 8x8 matrix containing the desired quantization values. The first element in the array must be an 8. A null pointer sets the null default. Get returns a pointer to a matrix containing the quantization values. A null pointer indicates the null default.

COMPRESSOR_NON_INTRA_QUANTIZATION_TABLE

Description	Set quantization matrix for non-I-frame compression.
Access	set / get
Type	Xil_unsigned8[64]
Values	1 - 255
Default	ND = null
Notes	Set by passing a pointer to an 8x8 matrix containing the desired quantization values. A null pointer sets the null default. Get returns a pointer to a matrix containing the quantization values. A null pointer indicates the null default.

COMPRESSOR_PATTERN

Description	A structure containing a string of length greater than 0 and an integer repeat count. The string sets the pattern of picture types (in display order) which will be employed by the compressor in all subsequent groups of pictures (GOPs). The repeat count determines the number of times this pattern occurs in a GOP; i.e., the number of pictures in a GOP is the length of the pattern string multiplied by the repeat count. However, if the COMPRESSOR_PATTERN attribute is reset, if a new quantization table is loaded via the COMPRESSOR_INTRA_QUANTIZATION_TABLE attribute or the COMPRESSOR_NON_INTRA_QUANTIZATION_TABLE attribute, or if the COMPRESSOR_INSERT_VIDEO_SEQUENCE_END attribute is set, the current GOP is terminated, and a new GOP is started on the next frame with a picture pattern that is synchronized with the beginning of

	the current pattern string.
Access	set/get

Type	typedef struct __XilMpeg1Pattern {
	  char* pattern;
	  Xil_unsigned32 repeat_count; } XilMpeg1Pattern;

Values	The pattern string must contain only the characters 'I', 'B', 'P', and 'D', which indicate Intra, Predicted, Bidirectional, and DC pictures. The repeat count must be greater than zero.
Default	ND = null
Notes	Set by passing a pointer to the pattern structure. A null string sets the null default. Get returns a pointer to the structure. If this is null, the null default is indicated. After a get which does not return null, it is the application's responsibility to free the pattern string and the structure storage.

COMPRESSOR_PEL_ASPECT_RATIO

Description Describes the pixel aspect ratio of the compressed image.

Access set/get

Type

```
typedef enum {
        NullDefault,
        Ratio_1_0,          /* 1.0        */
        Ratio_0_6735,       /* 0.6735     */
        Ratio_0_7031,       /* 0.7031     */
        Ratio_0_7615,       /* 0.7615     */
        Ratio_0_8055,       /* 0.8055     */
        Ratio_0_8437,       /* 0.8437     */
        Ratio_0_8935,       /* 0.8935     */
        Ratio_0_9157,       /* 0.9157     */
        Ratio_0_9815,       /* 0.9815     */
        Ratio_1_0255,       /* 1.0255     */
        Ratio_1_0695,       /* 1.0695     */
        Ratio_1_0950,       /* 1.0950     */
        Ratio_1_1575,       /* 1.1575     */
        Ratio_1_2015        /* 1.2015     */
} XilMpeg1PelAspectRatio;
```

Values The enumeration forms a discrete set of "likely" possibilities defined in the MPEG specification; they vary from .6375 to 1.2015.

Default ND = NullDefault

COMPRESSOR_PICTURE_RATE

Description	Describes the picture rate in frames per second of the image sequence to be compressed.
Access	set / get
Type	

```
typedef enum {
        NullDefault,
        Rate_23_976,          /* 23.976      */
        Rate_24,              /* 24.0        */
        Rate_25,              /* 25.0        */
        Rate_29_97,           /* 29.97       */
        Rate_30,              /* 30.0        */
        Rate_50,              /* 50.0        */
        Rate_59_94,           /* 59.94       */
        Rate_60               /* 60.0        */
} XilMpeg1PictureRate;
```

Values	The enumeration forms a discrete set corresponding to commonly available sources of digital or analog video, varying from 23.96 to 60.0.
Default	ND = NullDefault

COMPRESSOR_SLICES_PER_PICTURE

Description	Provide a suggestion to the compressor on how many slices to use in each picture.
Access	set / get
Type	int value
Values	1 - number of macroblocks in the picture
Default	ND = 0

Notes	Although the compressor is free to ignore this suggestion, setting this attribute to a high value may result in an inefficient use of the available bit rate.

COMPRESSOR_TIME_CODE

Description A time code that applies to the first picture (in display order) in the group of pictures (GOP) to be encoded. It is included to provide video time identification to applications.

Access set / get

Type

```
typedef struct __XilMpeg1TimeCode {
        Xil_boolean      drop_frame_flag;
        Xil_unsigned32   hours;
        Xil_unsigned32   minutes;
        Xil_unsigned32   seconds;
        Xil_unsigned32   pictures;
} XilMpeg1TimeCode;
```

Values The time code structure contains fields with integer values: hours (0-23), minutes (0-59), seconds (0-59), and picture number (0-59).

Default ND = null

Notes Set by passing a pointer to the time code structure. A null pointer sets the null default. Get returns a pointer to a structure containing the time information or null if the null default is set.

MPEG Decompression Attributes

DECOMPRESSOR_QUALITY

Description Provide a suggestion to the decompressor, enabling it to trade off reconstruction quality in exchange for an increase in decoding speed.

Access set / get

Type int value

Values Value can be between 1 and 100. A value of 100 sets the quality to maximum. A value of 1 sets the speed to its maximum and allows the quality to decrease to the minimum allowed by the decompressor. The decompressor is free to ignore this suggestion.

Default 100

Notes The MPEG decompressor may increase speed by such devices as decreasing the number of quantized coefficients that it uses in reconstruction, rounding motion vectors to integer values, etc.

DECOMPRESSOR_BROKEN_LINK

Description	Describes whether the B-pictures that precede the first I-picture in the GOP can be correctly decoded.
Access	get
Type	Xil_boolean
Values	FALSE - can be correctly decoded; TRUE - cannot be correctly decoded.
Default	FALSE
Notes	If this attribute is set to TRUE, it implies that the I or P picture from the previous group required to form the predictions is not available (presumably because it was removed as part of an editing process).

DECOMPRESSOR_CLOSED_GOP

Description	Describes whether the group of pictures is open or closed.
Access	get
Type	Xil_boolean
Values	FALSE - open group; TRUE - closed group.
Default	None
Notes	Closed groups can be decoded without using decoded pictures of the previous group for motion compensation. Open groups require such pictures to be available.

DECOMPRESSOR_FRAME_TYPE

Description Gives the picture type of the current picture in the group.

Access get

Type

```
typedef enum {
        I, P, B, D
}XilMpeg1FrameType
```

Values Values of the enumerated type are I, P, B, and D.

Default None

Notes The values 'I', 'B', 'P',and 'D' indicate Intra, Predicted, Bidirectional, and DC pictures.

DECOMPRESSOR_PEL_ASPECT_RATIO_VALUE

Description	Describes the pixel aspect ratio of the decompressed image.
Access	get

Type	float value
Values	The set of possible values forms a discrete set of "likely" possibilities defined in the MPEG specification; they vary from .6375 to 1.2015.
Default	1.0

DECOMPRESSOR_PICTURE_RATE_VALUE

Description	Describes the picture rate of the MPEG encoded image sequence in frames per second.
Access	get
Type	float value
Values	The set of possible values forms a discrete set corresponding to commonly available sources of digital or analog video, varying from 23.96 to 60.0.
Default	None

DECOMPRESSOR_TEMPORAL_REFERENCE

Description	Gives the number in the temporal reference field of the current picture in the group.
Access	get
Type	int value
Values	Between 0 and 1023
Default	None
Notes	This may be useful, because MPEG pictures are not transmitted in display order, but rather in the order in which the decoder needs to decode them.

DECOMPRESSOR_TIME_CODE

Description	A time code that applies to the first picture (in display order) in a group of pictures (GOP). It is included to provide video time identification to applications.
Access	get
Type	

```
typedef struct __XilMpeg1TimeCode {
        Xil_boolean      drop_frame_flag;
        Xil_unsigned32   hours;
        Xil_unsigned32   minutes;
        Xil_unsigned32   seconds;
        Xil_unsigned32   pictures;
} XilMpeg1TimeCode;
```

Values	The time code structure contains fields with integer values: hours (0-23), minutes (0-59), seconds (0-59), and picture number (0-59).
Default	None

EXAMPLES

The following example opens and closes an MPEG CIS using the XIL library.

```
XilSystemState State;
XilCis cis;
State = xil_open();
cis = xil_cis_create(State, "Mpeg1");

-- calls to MPEG-specific compression routines --

xil_cis_destroy(cis);
xil_close(State);
```

The following example sets an MPEG CIS attribute called *COMPRESSOR_SLICES_PER_PICTURE* to 3. Note that because this attribute is not a structure, it is not necessary to pass the address of this attribute when setting it.

```
XilCis cis;

xil_cis_set_attribute(cis,"COMPRESSOR_SLICES_PER_PICTURE", (void *) 3);
```

The following example returns the value of an MPEG CIS attribute called *COMPRESSOR_SLICES_PER_PICTURE.* Note that when getting an attribute, it is always necessary to pass the address.

```
Xil_unsigned32 slices;
XilCis cis;

xil_cis_get_attribute(cis, "COMPRESSOR_SLICES_PER_PICTURE", (void **) &slices);
```

NOTES

The **xil_cis_set_attribute**(3) and **xil_cis_get_attribute**(3) calls are used to modify the default behavior of a specific compressor. Generic attributes of compressors are set by individual function calls.

SEE ALSO

xil_cis_create(3), **xil_cis_get_attribute**(3), **xil_cis_get_bits_ptr**(3), **xil_compress**(3), **xil_decompress**(3), **xil_cis_has_frame**(3).

NAME

PhotoCD – Reader for Kodak(tm) Photo CD(tm) format

DESCRIPTION

Kodak Photo CD allows digital data generated by scanning 35-mm film to be encoded and stored on a compact disc. The XIL library supports the following Photo CD image resolutions:

BASE / 16	192 x 128 pixels
BASE / 4	384 x 256
BASE	768 x 512
4BASE	1536 x 1024
16BASE	3072 x 2048
64BASE	6144 x 4096

Images on Kodak Photo CD are stored in the XIL library's "photoycc" color space. The Photo CD reader returns images in this color space. To display or further process the images, you normally convert the images to an RGB color space, such as "rgb709," by calling **xil_color_convert**(3). Grayscale or "Black and White" versions of the images may be obtained by converting to "y709" or "ylinear."

Using the Photo CD Reader

To access images from Photo CD files, supply "ioSUNWPhotoCD" for the *devicename* argument in **xil_create_from_device**(3), and specify NULL for the *deviceObj* argument. After creating the device image, it may be used as a source to any XIL image operation. Because Photo CD is a read-only device, the device image created by this device handler is read-only. Trying to use the device image as a destination will generate an error.

Use **xil_get_device_attribute**(3) and **xil_set_device_attribute**(3) to get and set the Photo CD reader attributes, as described below. The Photo CD reader doesn't use or recognize associated device objects, so you cannot initialize attributes for it. You must create the device image first, then set its attributes.

PhotoCD Reader Attributes

The following paragraphs describe the attributes of the Photo CD reader. The enumerated types are defined in **xilPhotoCD.h.** Note that some attributes are "set / get" and others are "get-only." This is noted under the *Access* heading for each attribute.

FILEPATH

Description	Pathname for the desired image pack. Setting this attribute directs the library to use the image pack with the given pathname for the next use as an image source. This attribute does not need to be reset for each use of the image as a source, only when a different image is desired. There is no default pathname; trying to use the created device image before setting this attribute will cause an error to be generated.
Access	set / get
Type	char *

RESOLUTION

Description	Describes the size of the image to be obtained from the Photo CD. The default value is XIL_PHOTOCD_BASE, or 768 x 512 pixels. After the value has been set, the *FILEPATH* attribute may be changed without changing the desired resolution. Conversely, the resolution may be changed without resetting the path attribute.
Access	set / get

Type
```
typedef enum{
        XIL_PHOTOCD_16TH_BASE,
        XIL_PHOTOCD_4TH_BASE,
        XIL_PHOTOCD_BASE,
        XIL_PHOTOCD_4X_BASE,
        XIL_PHOTOCD_16X_BASE
        XIL_PHOTOCD_64X_BASE
} XilPhotoCDResolution;
```

MAX_RESOLUTION

Description	Describes the maximum size obtainable from this image pack. In some cases, not all image sizes are available within an image pack (This is sometimes done for pre-recorded Photo CDs). This attribute returns the maximum size which may be asked for using the *RESOLUTION* attribute. The value returned is one of the sizes described below.
Access	get-only

Type
```
typedef enum{
        XIL_PHOTOCD_16TH_BASE,
        XIL_PHOTOCD_4TH_BASE,
        XIL_PHOTOCD_BASE,
        XIL_PHOTOCD_4X_BASE,
        XIL_PHOTOCD_16X_BASE
        XIL_PHOTOCD_64X_BASE
} XilPhotoCDResolution;
```

ROTATION

Description	Describes the amount of rotation required to display the image in its proper orientation. The value returned is one of the enumeration constants shown below.
Access	get-only

```
Type            typedef enum{
                        XIL_PHOTOCD_CCW0,
                        XIL_PHOTOCD_CCW90,
                        XIL_PHOTOCD_CCW180,
                        XIL_PHOTOCD_CCW270
                } XilPhotoCDRotate;
```

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

The following example opens a Photo CD image, checking the image's ROTATION attribute so it can rotate the image, if necessary, before displaying it, and so the display window has the appropriate dimensions. The example also converts the image to the RGB 709 color space for display.

```
XilSystemState state;
XilImage ycc_photocd_image, rgb_photocd_image, photocd_image, display;
XilPhotoCDRotate rotation;
char *pathname = "my_photocd_image";

state = xil_open();

/* create the device image */
ycc_photocd_image = xil_create_from_device(state, "ioSUNWPhotoCD", NULL);

/* set the file name and desired image size */
xil_set_device_attribute(ycc_photocd_image, "FILEPATH", pathname);
xil_set_device_attribute(ycc_photocd_image, "RESOLUTION",
       (void*) XIL_PHOTOCD_BASE);

/* read the image's rotation attribute and image parameters */
xil_get_device_attribute(ycc_photocd_image, "ROTATION", (void**)&rotation);
xil_get_info(ycc_photocd_image, &width, &height, &nbands, &datatype);

/* create an image to prepare for color-space conversion */
rgb_photocd_image = xil_create(state, width, height, nbands,
    datatype);

/* Set color spaces to prepare for color-space conversion */
xil_set_colorspace(ycc_photocd_image,
    xil_colorspace_get_by_name(state, "photoycc"));
xil_set_colorspace(rgb_photocd_image,
    xil_colorspace_get_by_name(state, "rgb709"));
```

```
/* Convert the image's color space so it can be displayed */
xil_color_convert(ycc_photocd_image, rgb_photocd_image);

/* Set the image origin in preparation for a rotation */
xil_set_origin(rgb_photocd_image, width/2.0, height/2.0);

/* Based upon the Photo CD image's ROTATION attribute, open a
 * window with appropriate dimentions; for 90 and 270 degree
 * rotation, the image is rotated. Also, set the Photo CD
 * image's origin to its center.  */
if(rotation == XIL_PHOTOCD_CCW90 | |
   rotation == XIL_PHOTOCD_CCW270) {

/* ...code to open a window with dimensions height-by-width ...*/

      display = xil_create_from_window(state, xdisplay, xwindow);
      photocd_image = xil_create(state, height, width, nbands,
        datatype);
      xil_set_origin(photocd_image, height/2.0, width/2.0);
     }
else {

/* ...code to open a window with dimensions width-by-height ...*/

      display = xil_create_from_window(state, xdisplay, xwindow);
      photocd_image = xil_create(state, width, height, nbands,
        datatype);
      xil_set_origin(photocd_image, width/2.0, height/2.0);
    }

/* Set the radians for a rotation. Constant PI has be previously
 * defined and set equal to 3.14159 */
switch (rotation) {
  case XIL_PHOTOCD_CCW0:
    angle = 0; break;
  case XIL_PHOTOCD_CCW90:
    angle = PI * 0.5; break;
  case XIL_PHOTOCD_CCW180:
    angle = PI; break;
  case XIL_PHOTOCD_CCW270:
    angle = PI * 1.5; break;
}

/* Rotate the image, then return its origin to 0,0 */
xil_rotate(rgb_photocd_image, photocd_image, "nearest", angle);
```

```
        xil_set_origin(photocd_image, 0.0, 0.0);

        /* copy the image to the display */
        xil_copy(photocd_image, display);
```

NOTES

The **xil_set_device_attribute**(3) and **xil_get_device_attribute**(3) calls are used to modify the default behavior of specific device images. Generic attributes of images are set by individual function calls.

SEE ALSO

xil_color_convert(3), **xil_create_from_device**(3), **xil_open**(3).

NAME

xil_absolute – find the absolute value of pixels of an image

SYNOPSIS

#include <xil/xil.h>

void xil_absolute (XilImage *src***,**
        XilImage *dst***);**

DESCRIPTION

xil_absolute () performs a pixel-by-pixel abs() operation on *src* image and stores the result in the *dst* (destination) image.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Find the absolute value of pixels in *image1* and store the result in *dst* :

```
XilImage image1, dst;

xil_absolute(image1, dst);
```

NOTES

Source and destination images must be of the same data type and have the same number of bands. In-place operations are supported.

NAME

xil_add, xil_add_const – image addition operations

SYNOPSIS

#include <xil/xil.h>

void xil_add (XilImage *src1***,**
    XilImage *src2***,**
    XilImage *dst***);**

void xil_add_const (XilImage *src1***,**
    float **constants***,**
    XilImage *dst***);**

DESCRIPTION

xil_add () performs a pixel-by-pixel addition of the *src2* image to the *src1* image and stores the result in the *dst* (destination) image. If the result of the operation is out of range for a particular data type, the result is clamped to the minimum or maximum value for the data type. Results for XIL_BYTE operations, for example, are clamped to 0 if they are less than 0 and 255 if they are greater than 255.

xil_add_const () performs a pixel-by-pixel addition of the *constants* values to the *src1* image and stores the result in the *dst* (destination) image. For an n-band image, n float values must be provided, one per band. Pixel values are rounded and clipped according to the image data type.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Add *image2* to *image1* and store the result in *dst* :

```
XilImage image1, image2, dst;

xil_add(image1, image2, dst);
```

Add *constants* to 4-band *image1* and store the result in *dst* :

```
XilImage image1, dst;
float constants[4];

constants[0] = 1.0;
constants[1] = 1.0;
constants[2] = 1.0;
constants[3] = 0.0;
xil_add_const(image1, constants, dst);
```

NOTES

Source and destination images must be of the same data type and have the same number of bands. In-place operations are supported.

NAME

xil_affine – perform an affine transform on an image

SYNOPSIS

#include <xil/xil.h>

void xil_affine (XilImage *src***,**
    XilImage *dst***,**
    char **interpolation***,**
    float **matrix***);**

DESCRIPTION

This function performs an affine transform on an image. *src* is the source image handle. *dst* is the destination image handle. *interpolation* is a string that specifies the type of interpolation to be used. The supported interpolation types are *nearest* (nearest neighbor), *bilinear, bicubic,* and *general. matrix* is a six-entry floating point array that defines an arbitrary forward affine transform. This transform combines a scale, rotation, and translation. The order of the entries in the matrix are: a, b, c, d, tx, ty. The equations that generate the warp are as follows:

```
xd = a*xs + c*ys + tx
yd = b*xs + d*ys + ty
```

where *xs* and *ys* are points in the source image, and *xd* and *yd* are points in the destination image.

ROI Behavior

If an ROI (region of interest) is attached to the source image, it is used as a read mask and is transformed into the destination image's space, where it is intersected with the destination ROI (if there is one).

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Transform an image using nearest neighbor interpolation and the following affine transform matrix: {2.0, 0.0, 0.0, 2.0, 10.0, 10.0}. This transform matrix scales an image by 2.0 in width and height) and translates it by 10 pixels in both the *x* and *y* directions.

```
XilImage src, dst;
float matrix[6] = {2.0, 0.0, 0.0, 2.0, 10.0, 10.0};

xil_affine(src, dst, "nearest", matrix);
```

NOTES

The source and destination images to be transformed must have the same data type and number of bands. This operation cannot be performed in place.

SEE ALSO

xil_translate(3), **xil_rotate**(3), **xil_scale**(3), **xil_transpose**(3).

NAME

xil_and, xil_and_const – bitwise logical AND operations

SYNOPSIS

#include <xil/xil.h>

void xil_and (XilImage *src1***,**
        XilImage *src2***,**
        XilImage *dst***);**

void xil_and_const (XilImage *src1***,**
        unsigned int **constants***,**
        XilImage *dst***);**

DESCRIPTION

xil_and () performs a bitwise logical AND operation on each pixel of the *src2* (source) image with the *src1* and stores the results in the *dst* (destination) image.

xil_and_const () performs a bitwise logical AND operation on each pixel of the *src1* (source) image with the *constants* values and stores the results in the *dst* (destination) image. For an n-band image, n unsigned integers must be provided, one per band.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Bitwise logical AND *image2* and *image1* and store the result in *dst* :

```
XilImage image1, image2, dst;

xil_and(image1, image2, dst);
```

Bitwise logical AND 4-band *image1* and 4 constants and store the result in *dst:*

```
XilImage image, dst;
unsigned int constants[4];

constants[0] = 1;
constants[1] = 0;
constants[2] = 0;
constants[3] = 0;
xil_and_const(image, constants, dst);
```

NOTES

Source and destination images must be the same data type and have the same number of bands. In-place operations are supported.

NAME

xil_band_combine – interband linear combination operation

SYNOPSIS

#include <xil/xil.h>

void xil_band_combine (XilImage *src***,**
    XilImage *dst***,**
    XilKernel *matrix***);**

DESCRIPTION

This function performs the arbitrary interband linear combination of an image using the specified matrix. *src* is the source image handle. *dst* is the destination image handle. *matrix* is the floating point matrix used to perform the linear combination. The width of the matrix must be one larger than the number of bands in the source image. The height of the matrix must be equal to the number of bands in the destination image. Because the matrix can be of arbitrary size, this function can be used to produce a destination image with a different number of bands from the source image.

The destination image is formed by performing a matrix-multiply operation between the bands of the source image and the specified matrix. The extra column of values is a constant that is added after the matrix-multiply operation takes place. The matrix is implemented as an *XilKernel.* For a source pixel with N bands represented by (s0,s1,s2,...,sN-1), and a destination pixel with M bands represented by (d0,d1,d2,..., dM-1), the corresponding (N+1) x M matrix:

```
a00      a10      a20      ...      aN0
a01      a11      a21      ...      aN1
 ...
a0(M-1) a1(M-1) a2(M-1) ...      aN(M-1)
```

would give for the first element in the destination pixel:

d0 = a00s0 + a10s1 + a20s2 + ... + a(N-1)0s(N-1) + aN0

For example, the following 4x3 matrix would give a destination image equal to the source image:

```
1.0      0.0      0.0      0.0
0.0      1.0      0.0      0.0
0.0      0.0      1.0      0.0
```

This 5x1 matrix would select the second band of a 4 band image:

```
0.0      1.0      0.0      0.0      0.0
```

This 4x1 matrix would generate a single-band luminance image from an RGB image with the standard bgr memory format:

```
0.114    0.587    0.299    0.0
```

This 4x3 matrix would invert the second band of a 3-band image:

1.0	0.0	0.0	0.0
0.0	-1.0	0.0	255.0
0.0	0.0	1.0	0.0

Notice that the fourth column of this last matrix corresponds to the "constant" that is added after the multiply-add steps. It should be in the range appropriate for the source and destination data types.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

The following example generates a single-band image that is the normalized sum of all the bands of a three-band source image.

```
#include <xil/xil.h>
XilSystemState  State;
XilImage        src, dst;
XilKernel       matrix;
unsigned int    width = 4, height = 1;
float           *matrix_values = {0.333, 0.333, 0.333, 0.0}

State = xil_open();

matrix = xil_kernel_create(State, width, height, 0, 0, matrix_values);

/* create a dst image the same type as source, but only 1 band */
dst = xil_create(State, xil_get_width(src), xil_get_height(src),
          1, xil_get_datatype(src));

xil_band_combine(src, dst, matrix);
```

NOTES

The key pixel values for the *XilKernel* object are not used by **xil_band_combine** (), and are ignored.

SEE ALSO

xil_kernel_create(3)

NAME

xil_black_generation – adjust the amount of black to be added to or removed from a CMYK image

SYNOPSIS

#include <xil/xil.h>

void xil_black_generation (XilImage *src***,**
        XilImage *dst***,**
        float *black,*
        float *undercolor***);**

DESCRIPTION

This function adjusts the amount of black to be added to and removed from an image. Both *src* and *dst* are image handles to a 4-band CMYK image. *black* is the fraction of color that forms the K channel. *undercolor* represents the fraction of color taken away from each of the C, M, and Y channels.

Channels for each pixel are defined as follows:

black channnel	=	black ∗ (minimum of C, M, Y)
cyan channel	=	C - (undercolor ∗ (minimum of C, M, Y))
magenta channel	=	M - (undercolor ∗ (minimum of C, M, Y))
yellow channel	=	Y - (undercolor ∗ (minimum of C, M, Y))

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Adjust a CMYK image:

```
XilImage src, dst;

xil_black_generation(src, dst, 0.7, 0.5);
```

NOTES

It is assumed that all imported CMYK images are generated by using the same function for black generation and undercolor removal. In-place operations are supported.

SEE ALSO

xil_color_convert(3), **xil_set_colorspace**(3).

NAME

xil_blend – blend two images according to an alpha image

SYNOPSIS

#include <xil/xil.h>

void xil_blend (XilImage *src1***,**
        XilImage *src2***,**
        XilImage *dst***,**
        XilImage *alpha***);**

DESCRIPTION

This function blends two images according to an alpha image. For each pixel in the sources, the corresponding pixel in the alpha image provides a value that determines a linear combination of the source pixel values. The destination value is determined by this equation:

dst = (1.0 - normalize(alpha)) ∗ src1 + normalize(alpha) ∗ src2

src1 and *src2* are the source image handles. *dst* is the destination image handle. *alpha* is the alpha image handle.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Blend *src1* and *src2* according to *alpha* and put the result in *dst:*

```
XilImage src1, src2, dst, alpha;

xil_blend(src1, src2, dst, alpha);
```

NOTES

The source images and destination images must be the same type and have the same number of bands. The alpha image must be a single-band image and can be any of the supported data types. In-place operations are supported.

NAME

xil_cast – cast an image from one data type into another

SYNOPSIS

#include <xil/xil.h>

void xil_cast (XilImage *src***,**
  XilImage *dst***);**

DESCRIPTION

This routine casts an image of one data type into the data type specified by the *dst* (destination) image. When a data type with a lesser number of bits is cast into a data type with a greater number of bits, the destination pixel values are the *src* (source) image's pixel values padded with zeroes in the most significant bits. When a data type with a greater number of bits is cast into a data type with a lesser number of bits, the destination image's pixel values are a mask of the appropriate number of least significant bits of the source image's pixel values. To control the indices in the output image, pass the image through a lookup table rather than casting it.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Cast byte image *image1* into a bit image and store the result in *image2* :

```
XilSystemState state;
XilImage image1, image2;

image1 = xil_create(state, 512, 512, 3, XIL_BYTE);
image2 = xil_create(state, 512, 512, 3, XIL_BIT);
          .
          .
          .
xil_cast(image1, image2);
```

NOTES

Source and destination images must have the same number of bands.

SEE ALSO

xil_lookup_create(3).

NAME

xil_choose_colormap – choose a colormap that reasonably represents a full-color image

SYNOPSIS

#include <xil/xil.h>

XilLookup xil_choose_colormap (XilImage *src***,**
        unsigned int *size***);**

DESCRIPTION

This function creates and returns an *XilLookup* colormap with *size* entries to represent the full-color *src* (source) image. *size* specifies the number of colormap entries in the resulting *XilLookup* object. **xil_choose_colormap**() accepts only 3 banded XIL_BYTE source images.

RETURN VALUES

NULL	could not generate *XilLookup*

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Create a reasonable *XilLookup* with 216 colormap entries to represent the full-color source image:

```
XilImage src;
unsigned int cmap_size = 216;
XilLookup cmap;

cmap = xil_choose_colormap(src, cmap_size);
```

NAME

xil_cis_attempt_recovery – attempt recovery after an error occurs in a compressed image sequence

SYNOPSIS

#include <xil/xil.h>

void xil_cis_attempt_recovery (XilCis *cis***,**
  unsigned int *nframes***,**
  unsigned int *nbytes***);**

DESCRIPTION

This function is used to attempt recovery from a non-autorecoverable error that occurs during the playback of a compressed image sequence (CIS). *cis* is the input CIS in which an error occurred. *nframes* is the maximum number of frames ahead that will be scanned through to recover from a non-autorecoverable error. *nbytes* is the maximum number of bytes ahead that will be scanned through to recover from an error. If both of these values are zero, then the attempt at recovery will search forward as many bytes or frames as necessary. If *nframes* is non-zero and *nbytes* is zero, then the error recovery mechanism will attempt to search *nframes* ahead with its best approximation of exactly how many bytes that should be. If *nbytes* is non-zero and *nframes* is zero, the search will go through *nbytes*, regardless of the number of frames.

xil_cis_attempt_recovery () only needs to be called for non-autorecoverable errors. Consult **xil_cis_get_autorecover**(3) for details.

Both autorecoverable and non-autorecoverable errors are reported to the user through the error handling mechanism. The user decides whether to attempt recovery of a non-autorecoverable error.

If the error is recoverable, after reporting the error, the attribute *AUTO_RECOVER* (see **xil_cis_get_autorecover**(3)) is checked to determine whether to attempt recovery. If the attribute is set to TRUE, recovery is attempted.

Non-autorecoverable errors are handled similarly, except the *AUTO_RECOVER* attribute has no effect on how these errors are handled. When a non-autorecoverable error is detected, the CIS is marked invalid before the user is notified of the error. The CIS is marked CIS_READ_INVALID for decompression and CIS_WRITE_INVALID for compression (see **xil_cis_get_read_invalid**(3) and **xil_cis_get_write_invalid**(3)). Thus, if an error occurs in one of the decompression routines, then compression routines or **xil_cis_put_bits**(3) can still write into the CIS.

After a non-autorecoverable error has occurred, the user can validate the CIS in one of three ways: by calling **xil_cis_reset**(3), seeking to a valid frame, or asking XIL to attempt recovery using **xil_cis_attempt_recovery** (). If the user attempts to seek to a valid frame and the CIS cannot successfully complete the request, a seek error is generated.

To find out where the CIS is located after the call to **xil_cis_attempt_recovery** (), use **xil_cis_get_read_frame**(3) to get the bext approximation of the current *read_frame*, and **xil_cis_has_data**(3) to get the exact number of bytes left in the CIS. By checking and comparing the values returned by **xil_cis_has_data** () before and after calls to **xil_cis_attempt_recovery** (), you can determine the exact number of bytes that were searched through. It is also possible to determine the approximate number of frames that were skipped by checking and comparing the values returned by **xil_cis_get_read_frame**(3) before and after calls to **xil_cis_attempt_recovery** (). If **xil_cis_attempt_recovery** () succeeds, the CIS is returned to a VALID state. You can determine whether this function was successful by testing the state of the CIS. If you set the number of bytes or number of frames to check through to a low value, multiple calls to this function may be necessary.

CIS error recovery has been implemented so that **xil_cis_attempt_recovery** () can be called from the error handling function itself. If this function is called during the error handling function, the current decompress call will fail regardless of whether recovery was successful, the CIS will be marked VALID, and the next decompress call will succeed (unless another error is encountered).

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

In this example, when an error occurs, the error handler is called and the user gives the CIS permission to search indefinitely in an attempt to recover. If the attempt is unsuccessful, then **xil_cis_has_data**(3) fails, and the decompression loop is halted as if the video concluded.

```
/*
 * Example Error Recovery
 */

Xil_boolean
  my_error_handler( XilError error )
{

    XilCis cis;
    XilObject obj;

    /* If an XIL-CIS error occurred */
    if ( ( xil_error_get_category(error) == XIL_ERROR_CIS_DATA ) &&
         ( (obj = xil_error_get_object(error)) != NULL) &&
         ( xil_object_get_type(obj) == XIL_CIS ) ) {

      cis = (XilCis)obj;

      /* Has an error occured that we can attempt to
       *   recover from?  If so, attempt recovery.
```

```
               */
              if (xil_cis_get_read_invalid(cis)) {

                 xil_cis_attempt_recovery(cis, 0, 0);

                 /* If the CIS is now OK, we've handled it correctly. */
                 if (!xil_cis_get_read_invalid(cis))
                    return TRUE;
              }
          }
          return xil_call_next_error_handler(error);

        }

        main() {

           XilCis cis;
           XilSystemState state;
           XilImage image;
           XilImage displayimage;
           XilLookup lookup;

           if ( ( state = xil_open() ) == NULL ) {
              printf(" Couldn't initialize XIL\n");
              exit(1);
           }

           /* install error handler */
           xil_install_error_handler( state, my_error_handler );

           while(xil_cis_has_data(cis)) {
              xil_decompress(cis, image);
              xil_nearest_color(image, displayimage, lookup);
           }
        }
```

NOTES Occasionally, it is possible that error recovery may revalidate the CIS, but be off-sync from the number of frames that would have been in the CIS if the data had been correct. This can cause another error later, when the CIS reaches the end of the data inserted through the **xil_cis_put_bits**(3) call and then finds that the number of frames that it decoded from the data chunk is different than what the user said was in it.

SEE ALSO

xil_cis_get_autorecover(3), **xil_cis_get_read_invalid**(3), **xil_cis_get_write_invalid**(3), **xil_cis_get_read_frame**(3), **xil_cis_put_bits**(3), **xil_call_next_error_handler**(3), **xil_cis_reset**(3).

NAME

xil_cis_create – create a new compressed image sequence

SYNOPSIS

#include <xil/xil.h>

XilCis xil_cis_create (XilSystemState *system_state***,**
  char **compressorname***);**

DESCRIPTION

This function creates a new compressed image sequence (CIS). A CIS is a container that holds compressed images for a particular type of compression. Therefore, the CIS type must match the compression type. *system_state* is a handle to the object returned by **xil_open**(3) when it is invoked. *compressorname* is a string that provides the name of a compressor recognized by the XIL library.

If this function is successful, then a handle to an *XilCis* object is returned, which may be used in subsequent calls to xil_cis-routines. When the *XilCis* object is no longer needed, release the resources associated with the CIS by passing its handle to **xil_cis_destroy**(3).

The XIL library currently supports the following compressors: "Jpeg", "JpegLL", "Cell", "CellB", "faxG3", "faxG4", "Mpeg1", and "H261". Consult the following man pages for descriptions of these compressors and their attributes: **Jpeg**(3), **JpegLL**(3), **Cell**(3), **CellB**(3), **faxG3**(3), **faxG4**(3), **Mpeg1**(3), and **H261**(3).

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Open and close a JPEG CIS using the XIL library:

```
XilSystemState State;
XilCis cis;
State = xil_open();
cis = xil_cis_create(State, "Jpeg");

-- calls to JPEG-specific compression routines --

xil_cis_destroy(cis);
xil_close(State);
```

SEE ALSO

xil_open(3), **xil_close**(3), **xil_cis_destroy**(3), **xil_cis_flush**(3), **xil_cis_put_bits**(3), **xil_cis_reset**(3), **xil_cis_seek**(3), **xil_compress**(3), **xil_decompress**(3).

NAME

xil_cis_destroy – destroy a compressed image sequence

SYNOPSIS

#include <xil/xil.h>

void xil_cis_destroy (XilCis *cis***);**

DESCRIPTION

This function destroys a compressed image sequence, freeing resources associated with the *XilCis* structure. Any data that was inserted into the *XilCis* with **xil_cis_put_bits_ptr**(3) is not automatically freed, but the associated callback *done_with_data* is called.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Deallocate storage associated with a compressed image sequence:

```
XilCis cis;

xil_cis_destroy (cis);
```

SEE ALSO

xil_cis_create(3), **xil_cis_put_bits_ptr**(3).

NAME

xil_cis_flush – complete pending operations for a compressed image sequence

SYNOPSIS

#include <xil/xil.h>

void xil_cis_flush (XilCis *cis***);**

DESCRIPTION

This function instructs the compressor to complete any pending write operations for the compressed image sequence *cis*.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide*.

EXAMPLES

Flush a compressed image sequence:

```
XilCis cis;

xil_cis_flush( cis );
```

SEE ALSO

xil_compress(3)

NAME

xil_cis_get_attribute, xil_cis_set_attribute – get and set a compressor attribute

SYNOPSIS

#include <xil/xil.h>

int xil_cis_get_attribute (XilCis *cis,*
    char *∗attribute***,**
    void *∗∗value***);**

int xil_cis_set_attribute (XilCis *cis,*
    char *∗attribute,*
    void *∗data***);**

DESCRIPTION

xil_cis_get_attribute () returns the value of the *attribute* of the *cis* (the specified compressed image sequence).

xil_cis_set_attribute () sets *attribute* of *cis* to *data,* a generic pointer to the attribute value.

Available attributes are described on the specific man pages for the compressors and decompressors available with the XIL library, including the following: **Jpeg**(3) (baseline JPEG), **JpegLL**(3) (lossless JPEG), **Cell**(3) (image compression technology developed by Sun), **CellB**(3) (video compression technology developed by Sun), **faxG3**(3) and **faxG4**(3) (for CCITT Group 3 and Group 4 facsimile devices), **Mpeg1**(3) (basic MPEG video compression), **H261**(3) (CCITT Recommendation H.261, Video Codec for Audiovisual Services at px64 kbits / s).

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

This example sets a JPEG CIS attribute called *ENCODE_INTERLEAVED* to TRUE.

```
XilCis cis;

xil_cis_set_attribute(cis,"ENCODE_INTERLEAVED", (void ∗) TRUE);
```

This example returns the value of a JPEG CIS attribute called *ENCODE_INTERLEAVED.*

```
Xil_boolean encode_type;
XilCis cis;

xil_cis_get_attribute(cis, "ENCODE_INTERLEAVED", (void ∗∗) &encode_type);
```

NOTES

The **xil_cis_set_attribute** () and **xil_cis_get_attribute** () calls are used to modify the default behavior of a specific compressor. Generic attributes of compressors are set by individual function calls.

SEE ALSO **xil_compress**(3), **xil_cis_create**(3), **xil_choose_colormap**(3), **xil_decompress**(3), **xil_open**(3), **xil_cis_get_bits_ptr**(3), **xil_cis_get_compression_type**(3), **xil_cis_get_compressor**(3), **xil_cis_get_input_type**(3), **xil_cis_get_max_frames**(3), **xil_cis_get_output_type**(3), **xil_cis_get_start_frame**(3), **xil_cis_has_data**(3), **xil_cis_put_bits**(3), **xil_cis_reset**(3), **Jpeg**(3), **JpegLL**(3), **Cell**(3), **CellB**(3), **faxG3**(3), **fax**G4**(3), Mpeg1**(3), **H261**(3).

NAME

xil_cis_get_autorecover, xil_cis_set_autorecover – allow autorecovery after a CIS error occurs

SYNOPSIS

#include <xil/xil.h>

Xil_boolean xil_cis_get_autorecover (XilCis *cis***);**

void xil_cis_set_autorecover (XilCis *cis***,**
Xil_boolean *on_off***);**

DESCRIPTION

This function gives permission to the XIL CIS compression and decompression functions to attempt recovery if an autorecoverable bitstream error occurs. *cis* is the compressed image sequence (CIS) that is being compressed or decompressed. The default value returned by **xil_cis_get_autorecover** () is *FALSE* (or OFF), which indicates that autorecovery will not be attempted after a bitstream error occurs unless **xil_cis_set_autorecover** () is called to turn it ON.

Two types of bitstream errors can occur during decompression of a CIS: autorecoverable and non-autorecoverable. An autorecoverable error is one with a predefined method of recovery. A non-autorecoverable error requires user intervention for recovery. When a non-autorecoverable error is detected, the CIS is marked invalid before the user is notified of the error. If a CIS is marked CIS_READ_INVALID for decompression, no further operations can be performed on this CIS until it has been marked valid again. A bitstream error in CIS compression and decompression can occur in any action on a CIS that requires the CIS to decode the bitstream or change the current read frame.

Calling this routine for codecs that do not have autorecoverable errors (for example, Cell) will not affect these codecs.

After a non-autorecoverable error occurs, the user can revalidate the CIS in one of three ways: by calling **xil_cis_reset**(3) to remove any compressed data currently stored in the CIS, by calling **xil_cis_seek**(3) to seek to a valid frame, or by attempting recovery using **xil_cis_attempt_recovery**(3). If the user attempts to seek to a valid frame and the CIS cannot successfully complete the request, a seek error is generated.

See **xil_cis_attempt_recovery**(3) for further information on CIS error recovery.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

This example turns on auto-recovery:

```
XilCis cis;

xil_cis_set_autorecover(cis TRUE);
```

SEE ALSO

xil_cis_get_write_invalid(3), **xil_cis_attempt_recovery**(3), **xil_cis_seek**(3), **xil_cis_reset**(3).

NAME

xil_cis_get_bits_ptr – get compressed data from a compressed image sequence

SYNOPSIS

#include <xil/xil.h>

void* **xil_cis_get_bits_ptr (XilCis** *cis***,**
int *nbytes***,**
int *nframes***);**

DESCRIPTION

This function returns a generic pointer to data in a compressed image sequence. *cis* is the compressed image sequence that contains the compressed data for which a pointer is needed. *nbytes* indicates the number of bytes of data to which the generic pointer is pointing. *nframes* indicates the number of frames the compressed data represents.

The data pointed to is valid until one of the following routines is called, **xil_cis_get_bits_ptr** (), **xil_cis_reset**(3), **xil_compress**(3), or until the compressed image sequence is destroyed.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Extract the current information from a CIS and put it in a file.

```
XilCis cis;
char *data;
int nframes;
int nbytes;
FILE *f;

while (xil_cis_has_frame(cis)) {
        data = (char*)xil_cis_get_bits_ptr(cis, &nbytes, &nframes);
        fwrite(data, nbytes, 1, f);
}
```

SEE ALSO

xil_cis_create(3), **xil_cis_reset**(3), **xil_cis_put_bits_ptr**(3), **xil_compress**(3), **xil_cis_has_data**(3), **xil_cis_has_frame**(3).

NAME

xil_cis_get_by_name, xil_cis_get_name, xil_cis_set_name – get and set a compressed image sequence (CIS) object name

SYNOPSIS

#include <xil/xil.h>

XilCis xil_cis_get_by_name (XilSystemState *State***,**
  char **name***);**

char* xil_cis_get_name (XilCis *cis***);**

void xil_cis_set_name (XilCis *cis***,**
  char **name***);**

DESCRIPTION

Use these functions to assign names to CIS objects, and to retrieve CIS objects by name.

xil_cis_get_by_name () returns the handle to the CIS object with the specified name *name*. If such an object does not exist, NULL is returned. **xil_cis_get_by_name** () does not make a copy of the CIS object.

xil_cis_get_name () returns a copy of the specified CIS object's name. A call to **free** (3) should be used to free the space allocated by **xil_cis_get_name** (). If the specified CIS object has no name, NULL is returned.

xil_cis_set_name () sets the name of the specified CIS object to the one provided.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Allow a user to add images to a named CIS:

```
void add_image_to_cis(XilSystemState State, char* name, XilImage image);
{
  XilCis cis;

  cis = xil_cis_get_by_name (State, name);
  if (cis == NULL) {
    cis = xil_cis_create (State, "faxG3");
    xil_cis_set_name (cis, name);
  }
  xil_compress (image, cis);
  return;
}
```

NOTES

If you give two CIS objects the same name, it is not defined which CIS object will be retrieved by a call to **xil_cis_get_by_name** ().

NAME

xil_cis_get_compression_type – return the name of the type of the compressor

SYNOPSIS

#include <xil/xil.h>

char *xil_cis_get_compression_type (XilCis *cis***);**

DESCRIPTION

This function returns a character pointer to a string description of the type of the compressor for the *cis* (the specified compressed image sequence).

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

```
XilCis cis;
char *compression;

compression = xil_cis_get_compression_type(cis);
```

SEE ALSO

xil_compress(3), **xil_cis_get_compressor**(3).

NAME

xil_cis_get_compressor – return the name of a specific compressor

SYNOPSIS

#include <xil/xil.h>

char ∗xil_cis_get_compressor (XilCis *cis***);**

DESCRIPTION

This function returns a character pointer to a string description of a specific compressor for the *cis* (the specified compressed image sequence).

NOTES

This routine is very similar in operation to **xil_cis_get_compression_type**(3). In the future, there may be different types of *JPEG* compressions from different vendors, and this routine will be used to distinguish between them.

For example, any *cis* that returns a type of *JPEG* through a call to **xil_cis_get_compression_type**(3) should be able to decode a valid *JPEG* bitstream.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

```
XilCis cis;
char *compressor;

compressor = xil_cis_get_compressor(cis);
```

SEE ALSO

xil_compress(3), **xil_cis_get_compression_type**(3).

NAME

xil_cis_get_input_type – return the XilImageType that the CIS will accept for compression

SYNOPSIS

#include <xil/xil.h>

XilImageType xil_cis_get_input_type (XilCis *cis***);**

DESCRIPTION

This function returns the preferred image type that the *cis* (the specified compressed image sequence) will accept for compression. Unless a *cis* is documented as handling multiple input types, then this is the only type that the *cis* will accept. If *xsize, ysize,* or *nbands* are 0, then *cis* will currently accept images that vary in these dimensions.

Information about the image type that is not available when you first create a CIS may become available after your first call to the **xil_compress**(3) function. In other words, the values of *xsize* and *ysize* will never be zero after you call **xil_compress**(3).

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

```
XilCis cis;
XilImageType pref_type;
XilDataType cis_datatype;
unsigned int cis_xsize, cis_ysize, cis_nbands;

pref_type = xil_cis_get_input_type(cis);
xil_imagetype_get_info(pref_type, &cis_xsize, &cis_ysize,
                &cis_nbands, &cis_datatype);

printf("Preferred CIS has width=%d height=%d nbands=%d datatype=%d\n",
        cis_xsize, cis_ysize, cis_nbands, cis_datatype);
```

SEE ALSO

xil_compress(3), **xil_cis_create**(3).

NAME

xil_cis_get_max_frames, xil_cis_set_max_frames, xil_cis_get_keep_frames, xil_cis_set_keep_frames – get or set the upper limit on the number of compressed frames that a CIS should buffer

SYNOPSIS

#include <xil/xil.h>

int xil_cis_get_max_frames (XilCis *cis***);**

void xil_cis_set_max_frames (XilCis *cis***,**
    int *max_frames_to_buffer***);**

int xil_cis_get_keep_frames (XilCis *cis***);**

void xil_cis_set_keep_frames (XilCis *cis***,**
    int *frames_to_keep***);**

DESCRIPTION

xil_cis_set_max_frames () sets the upper limit on the number of compressed frames that the compressed image sequence (CIS) should buffer. A value of -1 means no limit. The default size depends on the compressor. The setting is a suggestion rather than a requirement, because some compression algorithms may not be able to function reasonably on an arbitrarily small buffer. An error occurs if a call to **xil_compress**(3), **xil_cis_put_bits**(3), or **xil_cis_put_bits_ptr**(3) results in more than *max_frames_to_buffer* frames in the CIS.

xil_cis_get_max_frames() retrieves the value set as the maximum number of compressed frames that the CIS will buffer at one time.

xil_cis_set_keep_frames() gets the number of frames before the read frame that the CIS should try to keep around. A value of -1 means no limit. In general, the number of keep frames should be smaller than the number of max frames.

Keeping the maximum number of frames in a buffer is a higher priority than keeping the maximum number of keep frames. Like **xil_cis_set_max_frames**(), the setting of the maximum number of keep frames is only a suggestion, because some decompression algorithms may not be able to function reasonably unless some set of previously read frames (such as key frames) exists in the CIS.

An error occurs when the number of frames between the start of the CIS and the read position falls below the set number of keep frames due to the addition of frames to the CIS and the CIS's attempt to keep the maximum number of frames in the entire CIS less than or equal to *max_frames.*

Seeking backward such that the number of frames before the read position becomes less than the desired keep frame value is not an error.

xil_cis_get_keep_frames() retrieves the value set as the maximum number of frames that the CIS should attempt to keep around.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

```
XilCis cis;
int mframes, kframes;
xil_cis_set_max_frames(cis , -1);
xil_cis_set_keep_frames(cis , 10);

mframes = xil_cis_get_max_frames(cis);

kframes = xil_cis_get_keep_frames(cis);
```

SEE ALSO **xil_compress**(3)

NAME

xil_cis_get_output_type – return the XilImageType produced by a compressor

SYNOPSIS

#include <xil/xil.h>

XilImageType xil_cis_get_output_type (XilCis *cis***);**

DESCRIPTION

This function returns the image type that the *cis* (the specified compressed image sequence) will produce upon decompression.

ERROR

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

```
XilCis cis;
XilImageType type;
int width;

type = xil_cis_get_output_type(cis);
width = xil_imagetype_get_width(type);
```

SEE ALSO

xil_decompress(3), **xil_imagetype_get_info**(3), **xil_get_imagetype**(3), **xil_cis_get_input_type**(3).

NAME

xil_cis_get_random_access – indicate whether a compressor supports random accessing of a CIS

SYNOPSIS

#include <xil/xil.h>

int xil_cis_get_random_access (XilCis *cis***);**

DESCRIPTION

This function returns a value that indicates whether a specified compressor supports random accessing of a compressed image sequence (CIS). If random accessing is supported, then **xil_cis_seek**(3) will be able to work for backwards seeks (forward seeks are always possible).

RETURN VALUES

TRUE	If the compressor supports random accessing of individual frames of the sequence
FALSE	If the compressor does not support random accessing of a CIS

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

```
XilCis cis;
 ...

if(xil_cis_get_random_access(cis) == TRUE) {
        printf("backwards seeks are enabled");
}
```

SEE ALSO

xil_compress(3), **xil_cis_seek**(3).

NAME

xil_cis_get_read_invalid – determine whether a CIS is able to be decompressed

SYNOPSIS

#include <xil/xil.h>

Xil_boolean xil_cis_get_read_invalid (XilCis *cis***);**

DESCRIPTION

This function determines whether a compressed image sequence (CIS) is able to be decompressed. *cis* is the CIS that is being decompressed. The default value returned by this routine is *FALSE,* which indicates that the CIS is valid and able to be decompressed. If a bitstream error occurs during decompression, this routine returns *TRUE,* indicating that the CIS was marked *CIS_READ_INVALID.*

Two types of bitstream errors can occur during decompression of a CIS: autorecoverable and non-autorecoverable. An autorecoverable error is one with a predefined method of recovery. A non-autorecoverable error requires user intervention for recovery. When a non-autorecoverable error is detected, the CIS is marked invalid before the user is notified of the error. If a CIS is marked *CIS_READ_INVALID* for decompression, no further operations can be performed on this CIS until it has been marked valid again.

After a non-autorecoverable error occurs, the user can revalidate the CIS in one of three ways: by calling **xil_cis_reset**(3) to remove any compressed data currently stored in the CIS, by calling **xil_cis_seek**(3) to seek to a valid frame, or by attempting recovery using **xil_cis_attempt_recovery**(3). If the user attempts to seek to a valid frame and the CIS cannot successfully complete the request, a seek error is generated.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Note that **xil_cis_get_read_invalid**() is not called until after the molecule runs. For information on molecules and deferred execution, consult the *XIL Programmer's Guide.*

```
XilCis cis;
XilImage image;
XilImage displayimage;
XilLookup lookup;

while(xil_cis_has_frame(cis)) {
   xil_decompress(cis, image);
   xil_nearest_color(image, displayimage, lookup);

   if (xil_cis_get_read_invalid(cis) == TRUE)
        printf(" There is a problem with this CIS.\n");
}
```

SEE ALSO

xil_cis_get_autorecover(3), **xil_cis_get_write_invalid**(3), **xil_cis_attempt_recovery**(3), **xil_cis_seek**(3), **xil_cis_reset**(3).

NAME

xil_cis_get_start_frame, xil_cis_get_read_frame, xil_cis_get_write_frame – obtain frame status attributes.

SYNOPSIS

#include <xil/xil.h>

int xil_cis_get_start_frame (XilCis *cis***);**

int xil_cis_get_read_frame (XilCis *cis***);**

int xil_cis_get_write_frame (XilCis *cis***);**

DESCRIPTION

In each of these routines, *cis* is the input compressed image sequence (CIS). Every frame in a CIS has a frame number associated with it. The beginning of the CIS is frame number zero. A CIS may have one or more frames buffered in memory. The *start_frame* is the index of the earliest buffered frame that still resides in the CIS. The *read_frame* is the index of the next frame that will be read by routines such as **xil_cis_get_bits_ptr**(3) or **xil_decompress**(3). The *write_frame* is the next frame that will be written. Routines such as **xil_cis_put_bits_ptr**(3) or **xil_compress**(3) add new frames immediately at this frame and increment the *write_frame* index each time they write a frame.

xil_cis_get_start_frame() returns the index, relative to the beginning of the CIS, to the first compressed image in the CIS.

xil_cis_get_read_frame() returns the index to the current read frame, the one that will be decompressed next.

xil_cis_get_write_frame() returns the index to the next frame that will be written. Thus, *write_frame* - 1 is the last complete frame in the CIS. If a partial or an unknown number of frames exist in the CIS because calls to **xil_cis_put_bits**()
or **xil_cis_put_bits_ptr**() have not yet been resolved, then the decompressor must parse the data to determine how many frames are in the CIS. This can make **xil_cis_get_write_frame**() potentially expensive.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

```
XilCis cis;

printf("Current Read Frame is %d\n",
        xil_cis_get_read_frame(cis));
```

SEE ALSO

xil_compress(3), **xil_decompress**(3), **xil_cis_seek**(3), **xil_cis_get_bits_ptr**(3), **xil_cis_put_bits_ptr**(3).

NAME

xil_cis_get_write_invalid – determine whether a CIS is able to continue to be compressed

SYNOPSIS

#include <xil/xil.h>

Xil_boolean xil_cis_get_write_invalid (XilCis *cis***);**

DESCRIPTION

This function determines whether compression is able to continue for a compressed image sequence (CIS). *cis* is the CIS that is being compressed. The default value returned by this routine is *FALSE*, which indicates that the CIS is valid and compression can continue. If a bitstream error occurs during compression, this routine returns *TRUE*, indicating that the CIS was marked *CIS_WRITE_INVALID*.

Two types of bitstream errors can occur during compression of a CIS: autorecoverable and non-autorecoverable. An autorecoverable error is one with a predefined method of recovery. A non-autorecoverable error requires user intervention for recovery. When a non-autorecoverable error is detected, the CIS is marked invalid before the user is notified of the error. If a CIS is marked *CIS_WRITE_INVALID* for compression, no further operations can be performed on this CIS until it is marked valid again.

After a non-autorecoverable error occurs, the user can revalidate the CIS in one of two ways: by calling **xil_cis_reset**(3) to remove any compressed data currently stored in the CIS, or attempting recovery using **xil_cis_attempt_recovery**(3).

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Determine if an error has occurred in the compression of a CIS:

```
XilCis cis;
XilImage src;

xil_compress( src, cis );

/* check to see if the cis is still valid. */
if (xil_cis_get_write_invalid(cis) == TRUE) {
  printf(" There is a problem with this CIS.\n");
}
```

SEE ALSO

xil_cis_get_autorecover(3), **xil_cis_get_read_invalid**(3), **xil_cis_attempt_recovery**(3), **xil_cis_reset**(3).

NAME

xil_cis_has_data, xil_cis_has_frame, xil_cis_number_of_frames – determine number of bytes or frames in a compressed image sequence

SYNOPSIS

#include <xil/xil.h>

int xil_cis_has_data (XilCis *cis***);**

Xil_boolean xil_cis_has_frame (XilCis *cis***);**

int xil_cis_number_of_frames (XilCis *cis***);**

DESCRIPTION

xil_cis_has_data () determines how many bytes of compressed data the compressed image sequence *cis* contains. This number reflects the number of bytes from the current read frame in the compressed image sequence (CIS) to the end of the CIS.

The number includes any bytes in an uncompleted frame at the end of a CIS. If the number of bytes is greater than zero, you can get a pointer to the data in the CIS by calling **xil_cis_get_bits_ptr**(3). However, you may not be able to read all of the data from the CIS at one time, because that data may not be in one contiguous buffer.

Also note that if all data has been retrieved from the CIS except for an incomplete frame at the CIS's end, **xil_cis_has_data**() returns a value greater than zero even though **xil_cis_get_bits_ptr**(3) will not be able to retrieve the data, because the last frame is not complete.

xil_cis_has_frame() returns TRUE if a complete frame exists at the read frame position, and returns FALSE otherwise. This routine can be used before calls such as **xil_decompress**(3) and **xil_cis_get_bits_ptr**(3) to test whether data is available for the desired operation. It is generally a better test than **xil_cis_has_data**() or **xil_cis_number_of_frames**() for determining the existence of data at the read frame position.

xil_cis_number_of_frames() determines how many complete frames of compressed data the compressed image sequence *cis* contains. This number reflects the number of frames from the current read position in the CIS to the last complete frame in the CIS. If a user inserts an unknown or partial number of frames in an *XilCis,* then the decompresser must parse the data to determine how many frames are in the *XilCis.* This can make **xil_cis_number_of_frames**() potentially expensive the first time it is called after a call to **xil_cis_put_bits**(3) or **xil_cis_put_bits_ptr**(3) and supplying a partial frame or an unknown number of frames.

RETURN VALUES

xil_cis_has_data() returns the number of bytes from the current read frame in the CIS to end of the CIS.

xil_cis_has_frame() returns TRUE if a complete frame exists at the read position; otherwise, FALSE.

xil_cis_number_of_frames() returns the number of frames from the current read frame in the CIS to end of the CIS.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

This example demonstrates that you can use any of three routines to determine if there are any frames in the CIS. Note that if all you are trying to do is determine if any frames are left in a CIS, then **xil_cis_has_frame**() is the preferred routine for accomplishing this.

The following loops extract all the bits between (and including) the read frame and the end of the CIS. In this example, nothing is done with the bits that are extracted. As it stands, if a partial frame exists at the end of the CIS, the **xil_cis_has_data**() loop never terminates.

```
XilCis cis;
char* data;
int nframes;
int nbytes;

while (xil_cis_number_of_frames(cis))
    data = (char *)xil_cis_get_bits_ptr(cis, &nbytes, &nframes);

while (xil_cis_has_data(cis))
    data = (char *)xil_cis_get_bits_ptr(cis, &nbytes, &nframes);

while (xil_cis_has_frame(cis))
    data = (char *)xil_cis_get_bits_ptr(cis, &nbytes, &nframes);
```

SEE ALSO

xil_cis_get_bits_ptr(3), **xil_cis_create**(3).

NAME

xil_cis_put_bits, xil_cis_put_bits_ptr – put compressed data into a compressed image sequence

SYNOPSIS

```
#include <xil/xil.h>

void xil_cis_put_bits (XilCis cis,
        int nbytes,
        int nframes,
        void *data);

void xil_cis_put_bits_ptr (XilCis cis,
        int nbytes,
        int nframes,
        void *data,
        XIL_FUNCPTR_DONE_WITH_DATA done_with_data);

typedef void (*XIL_FUNCPTR_DONE_WITH_DATA)(void *);
```

DESCRIPTION

xil_cis_put_bits() copies *nbytes* of compressed data representing *nframes* frames of uncompressed data into the compressed image sequence *cis.* Parameter *data* is a generic pointer to the data being copied into the compressed image sequence (CIS).

xil_cis_put_bits_ptr() puts *nbytes* of compressed data representing *nframes* frames of uncompressed data into the compressed image sequence *cis.* Parameter *data* is a generic pointer to the data being put into the CIS.

Unlike **xil_cis_put_bits**(), in **xil_cis_put_bits_ptr**() does not copy data into the CIS; instead, the CIS directly references the data pointed to by *data.* In this case, the application is responsible for ensuring that the data remains valid. The application may supply a routine *done_with_data()* that is called when the particular buffer is no longer needed by the CIS. The *done_with_data()* routine will also be called if the CIS is destroyed explicitly with **xil_cis_destroy**(3) or implicitly with **xil_close**(3). The application may supply NULL for the callback; in this case, the application is responsible for determining when particular buffers are no longer needed.

The *nframes* parameter is used to specify how many frames of uncompressed data the *nbytes* of compressed data represents. Used in this way, *nframes* must be an integer greater than zero. If the exact number of complete frames is not known, then *nframes* should be set to -1. This informs the CIS that the data being placed into it contains one or more complete frames.

If the data being put into the CIS may not represent an integer number of frames, then *nframes* should be set to 0. This informs the CIS that the data being placed into it may contain 0 or more frames, and that the last frame and / or the first frame represented in this data may not be complete.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Copy bitstream data that contains *frame_count* frames into an *XilCis:*

```
XilCis cis;
xil_cis_put_bits(cis, bytes, frame_count, data);
```

Copy bitstream data that contains an unknown number of complete frames (greater than or equal to 1 frame) into an *XilCis:*

```
XilCis cis;
xil_cis_put_bits(cis, bytes, -1, data);
```

Insert into an *XilCis* bitstream data that contains some number of frames in which the last and/or the first frame may or may not be complete:

```
XilCis cis;
xil_cis_put_bits_ptr(cis, bytes, 0, data, NULL);
```

NOTES

If error messages indicate that there is no more available free memory, try increasing swap space.

SEE ALSO

xil_cis_create(3)

NAME

xil_cis_reset – clears data in a compressed image sequence

SYNOPSIS

#include <xil/xil.h>

void xil_cis_reset (XilCis *cis***);**

DESCRIPTION

This function removes any compressed data currently stored in the specified compressed image sequence and sets the *cis* state parameters to their intitial values. *cis* is the compressed image sequence that contains the data to be cleared.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

```
XilCis cis;

xil_cis_reset(cis);
```

SEE ALSO

xil_compress(3)

NAME

xil_cis_seek – find a particular frame of compressed data in a compressed image sequence

SYNOPSIS

#include <xil/xil.h>

void xil_cis_seek (XilCis *cis***,**
  int *framenumber***,**
  int *relative_to***);**

DESCRIPTION

This function finds a particular frame of compressed data in a compressed image sequence (CIS). *cis* is the input compressed image sequence (CIS) in which you are looking for a frame of compressed data. *framenumber* is the number of frames by which the frame you are looking for is offset from the current frame, the beginning of the CIS, or the end of the CIS. *relative_to* indicates whether the offset mentioned above is relative to frame zero of the CIS (0), the current frame (1), or the end of the CIS (2).

Every frame in a CIS has a frame number associated with it; these frame numbers start at zero. Seeking from the beginning of the CIS implies that you are seeking relative to frame number zero and not necessarily the *start_frame* (the earliest buffered frame that still resides in the CIS). For more information see **xil_cis_get_start_frame**(3).

If the CIS you are looking in cannot be accessed randomly (see **xil_cis_get_random_access**(3)) and you are seeking a frame previous to the current *read_frame*, an error is generated.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide*.

EXAMPLES

Go to the 12th frame (from the beginning) of a compressed image sequence:

```
XilCis cis;

xil_cis_seek( cis, 12, 0 );
```

NOTES

The *framenumber* you are seeking must be within the CIS. Use the functions **xil_cis_get_start_frame**(3) and **xil_cis_get_write_frame**(3) to determine the legal range of frame numbers.

SEE ALSO

xil_cis_get_attribute(3), **xil_cis_get_start_frame**(3), **xil_cis_get_write_frame**(3), **xil_cis_get_random_access**(3).

NAME

xil_cis_sync – force any outstanding call to **xil_compress**(3) to complete when it would otherwise have been deferred

SYNOPSIS

#include <xil/xil.h>

void xil_cis_sync (XilCis *cis***);**

DESCRIPTION

xil_cis_sync() forces any outstanding calls to **xil_compress**(3) to complete. This forces the actual capture and compress to occur immediately instead of being deferred as part of an XIL molecule.

In order to execute multiple operations as a molecule, XIL defers the operations until they are needed. Thus, if a call to **xil_compress**() is part of a molecule, the capture and compression occurs when the deferred molecule is executed, not at the time that the **xil_compress**() function is called.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Measure the performance of a compress operation:

```
starttime= gmtime(NULL);          /* get the start time */
xil_compress(src,cis);            /* compress the image */
xil_cis_sync(cis);                /* force the compress to actually happen */
endtime= gmtime(NULL);            /* get the finish time */
```

NOTES

This function does not produce any semantic differences in the execution of the program.

NAME

xil_color_convert – converts an image from one color space to another

SYNOPSIS

#include <xil/xil.h>

void xil_color_convert (XilImage *src***,**
XilImage *dst***);**

DESCRIPTION

This function converts the data in the source image from the source image's color space to the destination image's color space. The color space is an attribute of each image. *src* is the source image's handle. *dst* is the destination image's handle. Neither the source nor the destination image can have a NULL color space.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

This example converts the data in *src* from *ycc601* colorspace to *rgblinear* colorspace:

```
XilSystemState State;
XilColorspace cspace1, cspace2;
XilImage src, dst;

cspace1 = xil_colorspace_get_by_name(State, "ycc601");
xil_set_colorspace(src, cspace1);

cspace2 = xil_colorspace_get_by_name(State, "rgblinear");
xil_set_colorspace(dst, cspace2);

xil_color_convert(src, dst);
```

NOTES

The source and destination images must be the same data type. The number of bands in an image must match its color space. In-place operations can be done by creating a child image consisting of the whole image and then assigning a different color space to the child image.

SEE ALSO

xil_colorspace_get_by_name(3), **xil_set_colorspace**(3), **xil_black_generation**(3).

NAME

xil_colorcube_create, xil_lookup_get_colorcube, xil_lookup_get_colorcube_info – operations on lookup tables used as colormap attributes of images

SYNOPSIS

#include <xil/xil.h>

XilLookup xil_colorcube_create (XilSystemState *State***,**
  XilDataType *input_type***,**
  XilDataType *output_type***,**
  unsigned int *nbands***,**
  short *offset***,**
  int *multipliers[]***,**
  unsigned int *dimensions[]***);**

Xil_boolean xil_lookup_get_colorcube (XilLookup *lookup***);**

Xil_boolean xil_lookup_get_colorcube_info (XilLookup *lookup***,**
  int ∗*multipliers***,**
  unsigned int ∗*dimensions***,**
  short ∗*origin***);**

DESCRIPTION

xil_colorcube_create () creates a lookup table that represents a colorcube. *input_type* is the data type of the input (either XIL_BIT, XIL_BYTE, or XIL_SHORT). *output_type* is the data type of the output. *nbands* is the number of bands of the colorcube. *offset* is the index of the first entry of the colorcube. *multipliers* is the distance between each color level in each dimension of the colorcube. These can be negative numbers to indicate decreasing color ramps rather than increasing color ramps. *dimensions* is a list of the sizes of each side of the colorcube.

xil_lookup_get_colorcube () returns TRUE or FALSE, depending on whether the specified lookup table is formatted as a colorcube.

xil_lookup_get_colorcube_info () returns TRUE or FALSE, depending on whether the specified lookup table is formatted as a colorcube. It also returns the *multipliers* and *dimensions* arrays for the colorcube. These arrays must be allocated by the user / application. The pointers to *multipliers* and *dimensions* may be NULL if that information is not needed.

origin is the index of the origin of the colorcube. In most cases, this should be the black pixel. If the *origin* is used as the starting index, then the *multipliers* can be used whether they have positive or negative values. The pointer may be NULL if the *origin* is not needed.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer’s Guide.*

EXAMPLES

Create an RGB colorcube with 4 shades of blue, 9 shades of green, and 6 shades of red that starts at index 16. When incrementing through the colors, blue changes most quickly, followed by greens, and then red.

```
static unsigned int dimensions[3]={4,9,6};
static int multipliers[3]={1,4,36};

xil_create_colorcube(State, XIL_BYTE, XIL_BYTE, 3, 16, multipliers, dimensions);
```

NOTES

A colorcube does not have to be three dimensional. It can have any number of dimensions from 1 to 65535. This makes it possible to have a colorcube for any color space.

Because the functions **xil_ordered_dither**(3), **xil_nearest_color**(3), and **xil_error_diffusion**(3) effectively push data backwards through a lookup table, the output of the colorcube must match the input to these functions, and the input of the colorcube must match the output of these functions.

SEE ALSO

xil_lookup_create(3), **xil_lookup_create_copy**(3), **xil_lookup_destroy**(3), **xil_lookup_convert**(3), **xil_lookup_set_values**(3).

NAME

xil_colorspace_get_by_name – get a colorspace object by its name

#include <xil/xil.h>

XilColorspace xil_colorspace_get_by_name (XilSystemState *State***,**
char ∗*name***);**

DESCRIPTION

This function retrieves color space objects by name. A number of predefined color spaces are created at the time of an **xil_open**(3) call. These color spaces can be retrieved by **xil_colorspace_get_by_name**().

Standard Color Spaces Provided

The XIL library creates a number of predefined colorspaces at the time of an **xil_open**(3) call. These color spaces include:

Color Space Name	*Description*
"rgb709"	Nonlinear RGB primaries as defined by CCIR Rec 709
"rgblinear"	Linearized RGB using primaries from CCIR Rec 709
"ycc709"	YCC as defined by CCIR Rec 709
"y709"	Luminance (black and white) from "ycc709"
"ylinear"	Linearized version of "y709"
"photoycc"	YCC color space defined by Kodak for PhotoCD
"ycc601"	YCC as defined by CCIR Rec 601
"y601"	Luminance from "ycc601"
"cmy"	Linear CMY, derived from "rgblinear"
"cmyk"	Linear CMYK, derived from "cmy" through undercolor removal

If an unsupported color space name is passed, **xil_colorspace_get_by_name**() returns NULL. Otherwise, a handle to the specified color space object is returned.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

```
XilSystemState State;
XilColorspace cspace;

State = xil_open();
cspace = xil_colorspace_get_by_name(State, "rgblinear");
```

NOTES

The set of standard objects is generated for each instantiation of an *XilSystemState.* If these standard objects are deleted, they become unavailable for the duration of the current XIL session.

If you give two color spaces the same name, it is not defined which color space will be retrieved by a call to **xil_colorspace_get_by_name**().

SEE ALSO

xil_color_convert(3), **xil_set_colorspace**(3), **xil_black_generation**(3), **xil_open**(3).

NAME

xil_compress – compress an image and write it to a compressed image sequence

SYNOPSIS

#include <xil/xil.h>

void xil_compress (XilImage *src***,**
XilCis *cis***);**

DESCRIPTION

This function compresses an image and writes the compressed data to a compressed image sequence (CIS). If the source image is a SEQUENTIAL device image, the function may compress multiple frames during a single execution.

src is the image (possibly a device image) containing the uncompressed data to be compressed. *cis* is the compressed image sequence to which the compressed data will be written. This object knows which compressor should be used to compress the data.

This function appends the compressed image at the CIS's current *write_frame* location, and then increments *write_frame.* Note that even after the **xil_compress** () operation occurs, the data for that frame is not guaranteed to be gotten by an **xil_cis_get_bits_ptr**(3) function, nor to be detectable by an **xil_cis_has_data**(3) operation, until **xil_cis_flush**(3) is called.

XIL Compressors

The XIL library provides the functions necessary to compress and decompress an image or sequence of images. This compressed data is stored in an object called a compressed image sequence (*XilCis*). A standard XIL compressor provides the following functions:

Compresses data and places it in a compressed image sequence (CIS).
Determines how much data a CIS contains.
Gets a pointer to compressed data in a CIS.
Empties a CIS.

A standard XIL decompressor provides the following functions:

Copies compressed data to a CIS.
Tells the decompressor to use the compressed data pointed to as the compressed data for a CIS.
Decompresses a frame of image data from a CIS and puts it in an image object (*XilImage*).
Locates a frame in a compressed image structure.
Determines the number of frames between the current frame and the end of the sequence, inclusive.

The XIL library supports a number of compression formats, including CCITT G3 / G4, JPEG, Cell and CellB, MPEG-1, and H.261 formats.

ROI Behavior

This function does not support source image ROIs.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Compress an image into a compressed image sequence:

```
XilImage src;
XilCis cis;
XilImageType type;
XilSystemState State;

type = xil_cis_get_input_type(cis);
src = xil_create_from_type(State, type);
/* generate the src image... */
xil_compress( src, cis );
```

NOTES

The *XilImageType* of the source image must match the input *XilImageType* of the CIS. Use **xil_cis_get_input_type**(3) to determine the required type.

SEE ALSO

xil_decompress(3), **xil_cis_get_bits_ptr**(3), **xil_cis_get_input_type**(3), **xil_cis_create**(3), **xil_cis_number_of_frames**(3), **xil_cis_flush**(3).

NAME

xil_convolve – convolve an image with a user-specified kernel

SYNOPSIS

#include <xil/xil.h>

void xil_convolve (XilImage *src***,**
    XilImage *dst***,**
    XilKernel *kernel***,**
    XilEdgeCondition *edge_condition***);**

DESCRIPTION

This function convolves an image with the specified kernel. *src* is the source image handle. *dst* is the destination image handle. *kernel* is a handle to an *XilKernel* structure that contains floating-point values.

edge_condition is an enumeration type that controls what happens when the convolution encounters the edge of an image. The three possible edge conditions are as follows:

XIL_EDGE_NO_WRITE	The edge of the destination image is not touched.
XIL_EDGE_ZERO_FILL	The edge of the destination image is set to zero.
XIL_EDGE_EXTEND	The edge of the source image is duplicated to provide information necessary for the convolution operation.

ROI Behavior

An ROI (region of interest) is used as a read mask for key pixels in the source image and as a write mask in the destination image. The convolve operation may access data outside a source ROI as long as the key pixel remains inside.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

For this example, a 2 x 2 kernel is created, and the key pixel is set to the lower right-hand corner of the kernel. Convolve the *src* image using the kernel, with the edge condition set to XIL_EDGE_ZERO_FILL.

```
XilSystemState State;
XilImage src, dst;
XilKernel kernel;
float data[4];

data[0] = data[1] = 0.5;
data[2] = data[3] = 0.0;
kernel = xil_kernel_create(State, 2, 2, 1, 1, data);

xil_convolve(src, dst, kernel, XIL_EDGE_ZERO_FILL);
```

NOTES

Source and destination images must be the same data type and have the same number of bands. The images need not have the same width and height. This operation cannot be performed in place.

SEE ALSO

xil_kernel_create(3), **xil_kernel_destroy**(3).

NAME

xil_copy – copy an image

SYNOPSIS

#include <xil/xil.h>

void xil_copy (XilImage *src*,
        XilImage *dst***);**

DESCRIPTION

This routine copies a *src* (source) image into a specified *dst* (destination) image. The source and destination images must be the same data type and have the same number of bands.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Copy *image1* into *tmp_image* :

```
XilImage image1, tmp_image;

xil_copy(image1, tmp_image);
```

NOTES

If overlapping but not coincident sibling images (children of the same parent) are specified as the source and destination, **xil_copy**( ) detects the overlap and correctly generates the destination image. All other operations generate a warning message under these conditions and have undefined results, as discussed in **xil_create_child**(3).

SEE ALSO

xil_copy_pattern(3), **xil_copy_with_planemask**(3).

NAME

xil_copy_pattern – replicate the source image into the destination image

SYNOPSIS

#include <xil/xil.h>

void xil_copy_pattern (XilImage *src*,
    XilImage *dst***);**

DESCRIPTION

This routine replicates the source image into the destination image. *src* is the source image handle. *dst* is the destination image handle.

For example, if the the size of the source image is 64 x 64 and the size of the destination image is 256 x 128, then the destination image will have (256 / 64) ∗ (128 / 64) = 8 copies of the source image. The size of the destination image does not have to be an even multiple of the size of the source image.

ROI Behavior

The source image ROI is repeated to be the same size as the destination image before intersection with the destination ROI.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Replicate the source image into the destination image:

```
XilImage src, dst;
xil_copy_pattern(src, dst);
```

NOTES

Source and destination images must be the same type and number of bands. In-place operations are not supported.

SEE ALSO

xil_copy(3)

NAME

xil_copy_with_planemask – using a plane mask, copy a source image into a destination image

SYNOPSIS

#include <xil/xil.h>

void xil_copy_with_planemask (XilImage *src***,**
    XilImage *dst***,**
    unsigned int *planemask[]***);**

DESCRIPTION

xil_copy_with_planemask () copies a *src* (source) image into a specified *dst* (destination) image, using a *plane mask* to specify which source-image planes (bits) are copied.

Each pixel in the destination image is defined by the following operation:

dst = (dst & ˜mask) | (src & mask)

Here, *dst* is the destination image, *mask* is the plane mask, and *src* is the source image. Thus, if the plane-mask bit is "on," the copy overwrites the corresponding bit in the destination image; otherwise, the bit in the destination image is unchanged.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Copy the low order bit of *src1* into the *dst* low order bit. Copy the high order seven bits of *src2* into the *dst* high order seven bits:

```
XilImage src1, src2, dst;
unsigned int planemask1[1], planemask2[1];

planemask1[0] = 0x1;
planemask2[0] = 0xfe;

xil_copy_with_planemask(src1, dst, planemask1);
xil_copy_with_planemask(src2, dst, planemask2);
```

NOTES

The plane mask must be an array of unsigned integers. The number of array elements must match the number of image bands; each array element specifies the plane mask for the corresponding band in the destination. Both the source and destination images must have the same type and number of bands. Standard ROI and in-place operations are supported.

When using a plane mask for copying an image to the display, the window's depth is the upper limit on the number of meaningful bits you can set in the plane mask, and you must manipulate the colormap to get a reasonable display.

SEE ALSO **xil_copy**(3), **xil_copy_pattern**(3).

NAME

xil_create – create an image

SYNOPSIS

#include <xil/xil.h>

XilImage xil_create (XilSystemState *State***,**
    unsigned int *width***,**
    unsigned int *height***,**
    unsigned int *nbands***,**
    XilDataType *datatype***);**

DESCRIPTION

This routine creates an image with the specified dimensions and data type. *width* is the width (extent in *x*) of the image. *height* is the height (extent in *y*) of the image. *nbands* is the number of bands in the image. *datatype* is the data type of the image, which can be one of the following enumeration constants of type XilDataType:

XIL_BIT	1-bit
XIL_BYTE	unsigned 8-bit
XIL_SHORT	signed 16-bit

If the function is successful, a handle to the image is returned. This is not the same as a pointer to the data, which is only available when the image is exported.

ROI Behavior

The default ROI is NULL. If an ROI is NULL, operations are performed on the entire image.

XIL Images

The primary objects in the XIL world are images. Each dimension of an image - width, height, or number of bands - may be as great as 65535 (a limitation of the XIL library), except that the overall size of an image is limited by available resources.

Three data precisions are supported: 1-bit, 8-bit unsigned, and 16-bit signed per data element.

The exposed attributes associated with images are *height, width, nbands* (number of bands -- number of distinct data elements per pixel), *datatype* (sample type -- precision of a single data element), color space, and image origin. You can get *height, nbands,* and *datatype* with **xil_get_info**(3) amd **xil_get_origin**(3). Note that the origin at creation time is the upper left corner of the image. Also note that an image's color space is NULL upon creation.

The XIL library currently has no provision for direct operation on images with bands of different data types or different dimensions. This implies no direct support for 4:1:1 or 4:2:2 data.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Create a 480x640 8-bit image with 3 bands, which can contain 8-bit unsigned data:

```
XilSystemState State;
XilImage image;
image = xil_create(State, 480, 640, 3, XIL_BYTE);
```

NOTES

The data associated with the image is not automatically zeroed. Use **xil_set_value**(3) to do this.

SEE ALSO

xil_create_child(3), **xil_create_copy**(3), **xil_create_from_device**(3), **xil_create_from_type**(3), **xil_create_from_window**(3), **xil_destroy**(3), **xil_set_roi**(3), **xil_get_roi**(3), **xil_get_info**(3), **xil_set_value**(3), **xil_get_origin**(3), **xil_set_colorspace**(3).

NAME

xil_create_child – create a child image

SYNOPSIS

#include <xil/xil.h>

XilImage xil_create_child (XilImage *src***,**
    unsigned int *xstart***,**
    unsigned int *ystart***,**
    unsigned int *width***,**
    unsigned int *height***,**
    unsigned int *startband***,**
    unsigned int *numbands***);**

DESCRIPTION

This routine creates a new (child) reference to the existing data. Modifications to the child image affect the parent data. *xstart* is the horizontal offset in pixels from the upper-left corner of the source image to the upper-left corner of the subimage. *ystart* is the vertical offset in pixels from the upper-left corner of the source image to the upper-left corner of the subimage. *width* is the width of the subimage in pixels. *height* is the height of the subimage in pixels. *startband* is the offset in bands, starting from the first band, to the first band in the subimage. *numbands* is the number of bands in the subimage.

The color space of the child image is set to that of the parent image if the number of bands in the child is the same as that of the parent. Otherwise, the color space is set to NULL. The origin of the child image is initialized to (0.0, 0.0).

Note that this function does not create a copy of the data, only a reference to it.

ROI Behavior

The default ROI is NULL. If an ROI is NULL, operations are performed on the entire (child) image. The parent image's ROI and origin are ignored by the child.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Create a 512 x 512 5-band, 16-bit image. Then create a 100 x 100 child image that begins at offset (200, 250) comprising the middle 3 bands:

```
XilImage image, child_image;

image = xil_create(512, 512, 5, XIL_SHORT);
child_image = xil_create_child (image, 200, 250, 100, 100, 1, 3);
```

NOTES

If overlapping but not coincident sibling images (children of the same parent) are specified as the source and destination for an operation, the operation is performed. However, the library generates a warning message, and the results of such an operation are undefined. For an exception to this behavior, see **xil_copy**(3).

SEE ALSO

xil_create(3), **xil_create_copy**(3), **xil_create_from_device**(3), **xil_create_from_type**(3), **xil_create_from_window**(3), **xil_destroy**(3), **xil_set_roi**(3), **xil_get_roi**(3), **xil_get_parent**(3).

NAME

xil_create_copy – create a new image with a copy of the source's data

SYNOPSIS

#include <xil/xil.h>

XilImage xil_create_copy (XilImage *src***,**
  unsigned int *xstart***,**
  unsigned int *ystart***,**
  unsigned int *width***,**
  unsigned int *height***,**
  unsigned int *startband***,**
  unsigned int *numbands***);**

DESCRIPTION

This routine creates a new image with its own copy of the source's data. *xstart* is the horizontal offset in pixels from the upper-left corner of the source image to the upper-left corner of the subimage. *ystart* is the vertical offset in pixels from the upper-left corner of the source image to the upper-left corner of the subimage. *width* is the width of the subimage in pixels. *height* is the height of the subimage in pixels. *startband* is the offset in bands, starting from the first band, to the first band in the subimage. *numbands* is the number of bands in the subimage.

Copies of images have the same XilVersion number as the original image. The name of a copy is initially empty (NULL).

ROI Behavior

The default ROI is NULL. If an ROI is NULL, operations are performed on the entire image. The ROI and the origin of the source image are ignored in the copy operation.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Create a 512 x 512 5-band, 16-bit image. Then copy a 100 x 100 image that begins at offset (200, 250) comprising the middle 3 bands into a new image:

```
XilImage image1, image2;

image1 = xil_create(512, 512, 5, XIL_SHORT);
image2 = xil_create_copy (image1, 200, 250, 100, 100, 1, 3);
```

SEE ALSO

xil_create(3), **xil_create_child**(3), **xil_create_from_device**(3), **xil_create_from_type**(3), **xil_create_from_window**(3), **xil_destroy**(3), **xil_set_roi**(3), **xil_get_roi**(3).

NAME

xil_create_from_type – create an image from an XilImageType object

SYNOPSIS

#include <xil/xil.h>

XilImage xil_create_from_type (XilSystemState *State*,
    XilImageType *imagetype***);**

DESCRIPTION

This routine creates an image from an *XilImageType* object. All the parameters needed to create the image are contained within the *XilImageType* object. An *XilImageType* object describes the characteristics of an image that will be generated (or expected) by a particular device (for example, a frame grabber or an output device). The characteristics of an *XilImageType* object are *xsize, ysize, nbands, datatype,* and *colorspace.* You obtain an *XilImageType* object from a call to **xil_cis_get_output_type**(3) or **xil_cis_get_input_type**(3). You can use an *XilImageType* object to create images (via the **xil_create_from_type** () call) that will be compatible with a given device, compressor, and so on. The origin of the image is initialized to (0.0, 0.0).

ROI Behavior

The default ROI is NULL. If an ROI is NULL, operations are performed on the entire image.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Create an image of the appropriate type to decompress a CIS into:

```
XilSystemState state;
XilImageType imagetype;
XilImage image;
XilCis cis;

imagetype = xil_cis_get_output_type (cis);
image = xil_create_from_type ( state, imagetype);
```

NOTES

The data associated with the image is not automatically zeroed. Use **xil_set_value**(3) to do this.

SEE ALSO

xil_create(3), **xil_create_copy**(3), **xil_create_from_device**(3), **xil_create_from_window**(3), **xil_destroy**(3), **xil_set_roi**(3), **xil_get_roi**(3), **xil_cis_get_output_type**(3), **xil_cis_get_input_type**(3), **xil_get_imagetype**(3).

NAME

xil_create_from_window, xil_create_from_device – create device images

SYNOPSIS

#include <xil/xil.h>

XilImage xil_create_from_window (XilSystemState *State*,
        Display ∗*display*,
        Window *window***);**

XilImage xil_create_from_device (XilSystemState *State*,
        char ∗*devicename*,
        XilDevice *deviceObj***);**

DESCRIPTION

These routines create images that are tied to particular devices. They allow X windows and various image input and output devices to be treated as if they were ordinary XIL images. After an image is created with the routines, the image can be read from the device or written to it by using the device as the source or destination of an image-processing operation.

xil_create_from_window () creates an image associated with the specified X window. Images can then be copied to this image for display. The default origin for images created with this function is (0.0, 0.0), and the default ROI is NULL.

xil_create_from_device () creates an image associated with the device named *devicename.* The parameter *deviceObj* is the handle to the device object associated with this device type; the device object is created with the **xil_device_create**(3) function and is used to store device-initialization values. The device object is used only for device types that require that one or more device attributes be initialized before the device image is created. If the device doesn't require attribute initialization, the device object isn't recongnized by the device type and you must pass NULL for the *deviceObj* parameter. The group that writes the device handler must indicate whether the device requires attribute initialization.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Create an XIL display image and copy it to a display image.

```
XilSystemState State;
XilImage display_image = NULL;
XilImage image0 = NULL;
Display ∗display;
Window window;

/∗ Create an XIL display image ∗/
display_image = xil_create_from_window(State, display, window);

/∗ Copy image0 to display image ∗/
```

xil_copy(image0,display_image);

NOTES

As with standard images, device images can have origins, color spaces, and so on. Subsets of device images can be referenced or written using regions of interest or child images.

To resize a window that contains an *XilImage,* destroy the *XilImage* attached to the window, resize the window, wait for a *ConfigureNotify* event to ensure the **XResizeWindow** () is complete, then call **xil_create_from_window** () to recreate the image in the new window size.

You cannot use an X window's *backing_store* attribute to maintain an image in the window when the window is obscured or unmapped (see the *Xlib Programming Manual).* Thus, your code should always check for an *Expose* event and take the appropriate measures for displaying the image again when the window is exposed.

SEE ALSO

xil_get_device_attribute(3), **xil_set_device_attribute**(3), **xil_get_readable**(3), **xil_get_writable**(3), **xil_device_create**(3), **xil_device_set_value**(3), **xil_device_destroy**(3).

NAME

xil_decompress – decompress an image from a compressed image sequence

SYNOPSIS

#include <xil/xil.h>

void xil_decompress (XilCis *cis***,**
        XilImage *dst***);**

DESCRIPTION

This function decompresses the current read frame in a compressed image sequence (CIS) and puts its output into an image object. It also increments the CIS's current read frame. *cis* is the input compressed image sequence. *dst* is the output *XilImage.* If the function is successful, an image from the CIS will be decompressed into the destination.

The XIL library supports a number of compression formats, including CCITT G3 / G4, JPEG, MPEG-1, H.261, Cell, and CellB.

ROI Behavior

A region of interest (ROI) is associated with each image. The default ROI encompasses the entire image. As a destination attribute, the ROI functions as a "write mask" for the destination image. Consult **xil_compress**(3) for more information.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Decompress the current read frame of a compressed image sequence:

```
XilCis cis;
XilImage dst;
XilImageType type;
XilSystemState State;
type = xil_cis_get_output_type(cis);
dst = xil_create_from_type(State, type);
while (xil_cis_has_frame(cis))
  xil_decompress(cis, dst);
```

NOTES

The data type and number of bands of the destination image must match the attributes of the images that are stored in compressed image sequence. Use **xil_cis_get_output_type**(3) to get a CIS's image type.

SEE ALSO

xil_compress(3), **xil_cis_has_frame**(3), **xil_cis_put_bits**(3), **xil_cis_put_bits_ptr**(3), **xil_cis_get_output_type**(3).

NAME

xil_destroy – destroy an image

SYNOPSIS

#include <xil/xil.h>

void xil_destroy (XilImage *image***);**

DESCRIPTION

This routine destroys an image, freeing the resources associated with the image structure. It also deallocates the memory used to store image data if that memory was allocated by XIL. If the image has child images allocated with it, they are also destroyed.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Destroy an image:

```
XilImage image;

xil_destroy (image);
```

NOTES

The user is responsible for freeing memory that has been assigned to an image via an **xil_set_memory_storage**(3) call.

Referencing an image after it has been destroyed (including any children that have been automatically destroyed) is an error that may cause problems potentially severe enough to cause a core dump.

If you create an XIL display image on an X display, you *must* destroy that image *before* calling **XCloseDisplay**(). Calling **XCloseDisplay**() before calling **xil_destroy**() will make **xil_destroy**() work improperly.

SEE ALSO

xil_create(3), **xil_create_child**(3), **xil_create_from_type**(3), **xil_create_copy**(3), **xil_create_from_window**(3), **xil_create_from_device**(3), **xil_set_memory_storage**(3).

NAME

xil_device_create, xil_device_destroy – create or destroy a device object

SYNOPSIS

#include <xil/xil.h>

XilDevice xil_device_create (XilSystemState *State*,
        char ∗*device***);**

void xil_device_destroy (XilDevice *deviceObj***);**

DESCRIPTION

xil_device_create () creates a device object and associates it with a particular device type; the object is used to store initialization attributes for its associated device. *State* is the XIL system state, and *device* is the name of the associated device type. The device name must be provided by the group that writes the device handler.

A device object is associated with a particular device type and cannot be associated with a different device type. Its only use is to initialize device attributes when you call the **xil_create_from_device**(3) function to create the device image. Device objects are particularly useful for storing interdependent attributes that must be simultaneously set for a device, or for setting attributes that require a substantial memory allocation.

xil_device_destroy () destroys the specified device object. Its only parameter is the handle to the device object.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Create a device object associated with the device "my_device":

```
XilSystemState State;
XilDevice deviceObj;

deviceObj = xil_device_create (State, "my_device");
```

NOTES

A device object cannot be used to adjust a device image's attributes after the image is created; **xil_set_device_attribute**(3) does that. However, after using the device object to create one device image, you can use the same object to store different initialization attributes, then use the modified device object when you create another device image of the same type.

Devices that don't require attribute initialization typically don't recognize or support device objects. For these devcies, you can't use a device object to set attributes.

SEE ALSO

xil_device_set_value(3), **xil_create_from_device**(3), **xil_set_device_attribute**(3).

NAME

xil_device_set_value – stores device-initialization values in a device object

SYNOPSIS

#include <xil/xil.h>

void xil_device_set_value (XilDevice *deviceObj***,**
    char ∗ *attribute***,**
    void ∗*value***);**

DESCRIPTION

xil_device_set_value () stores *attribute* and *value* in the device object *deviceObj. attribute* is the name of the attribute you want to set and *value* is the attribute's value. Attribute names and their possible values are defined by the group that writes the device handler. Only attributes the device understands should be set on the device object; otherwise an error is generated.

You can store in the object as many attributes and values as needed to derive all required initialization attributes for the device. Make a separate function call for each attribute.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Create a device object and pass it as an argument on the function call that creates its associated device image:

```
XilSystemState State;
XilDevice deviceObj;
XilImage dev_image;
int new_value = 255;

deviceObj = xil_device_create (State, "device");

xil_device_set_value (deviceObj, "ATTRIBUTE_1",
                        (void∗) new_value);
xil_device_set_value (deviceObj, "ATTRIBUTE_2",
                        (void∗) new_value);

dev_image = xil_create_from_device (State, "device", deviceObj);
```

NOTES

Because attributes and their associated values may reference data in the application's data space, any data associated with an XilDevice object must remain valid while the device object references it.

SEE ALSO

xil_device_create(3), **xil_create_from_device**(3), **xil_set_device_attribute**(3).

NAME

xil_dithermask_create, xil_dithermask_create_copy, xil_dithermask_destroy – create and destroy dither mask objects

SYNOPSIS

```
#include <xil/xil.h>

XilDitherMask xil_dithermask_create ( XilSystemState State,
        unsigned int width,
        unsigned int height,
        unsigned int nbands,
        float *data);

XilDitherMask xil_dithermask_create_copy ( XilDitherMask mask);

void xil_dithermask_destroy ( XilDitherMask mask);
```

DESCRIPTION

These routines create and destroy the *XilDitherMask* objects used in the **xil_ordered_dither**(3) operation.

xil_dithermask_create() creates an *XilDitherMask* object of the specified size with the specified data. *width* is the width of the dither mask in pixels. *height* is the height of the dither mask in pixels. *nbands* is the number of bands in the dither mask. *data* is a pointer to the data to be stored in the dither mask.

xil_dithermask_create_copy() creates and returns a copy of the specified dither mask. The name of a copy is initially empty (NULL).

xil_dithermask_destroy() destroys the specified dither mask.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Create a 4x4 1-band dither mask:

```
XilSystemState State;
unsigned int width=4, height=4, nbands=1;
XilDithermask dithermask;
float data[] =      {
                          0.0,     0.5,     0.125,   0.625,
                          0.75,    0.25,    0.875,   0.375,
                          0.1875,  0.6875,  0.0625,  0.5625,
                          0.9375,  0.4375,  0.8125,  0.3125
                    };

dithermask = xil_dithermask_create (State, width, height, nbands, data);
```

SEE ALSO

xil_dithermask_get_height(3), **xil_dithermask_get_by_name**(3), **xil_ordered_dither**(3).

NAME

xil_dithermask_get_by_name, xil_dithermask_get_name, xil_dithermask_set_name – get and set a dither mask object name and get the handle of a dither mask

SYNOPSIS

#include <xil/xil.h>

XilDitherMask xil_dithermask_get_by_name (XilSystemState *State***,**
    char ∗*name***);**

char∗ **xil_dithermask_get_name (XilDitherMask** *dithermask***);**

void xil_dithermask_set_name (XilDitherMask *dithermask***,**
    char ∗*name***);**

DESCRIPTION

Use these functions to assign names to dither mask objects, to retrieve dither mask objects by name, and to read the names of dither masks. For example, some predefined dither masks are created by an **xil_open**(3) call. These dither masks can be retrieved by **xil_dithermask_get_by_name**().

xil_dithermask_get_by_name()**returns**the**handle**to *name.* If such a dither mask does not exist, NULL is returned. **xil_dithermask_get_by_name**() does not make a copy of the dither mask.

xil_dithermask_get_name() returns a copy of the specified dither mask's name. A call to **free** (3) should be used to free the space allocated by **xil_dithermask_get_name**(). If the specified dither mask has no name, NULL is returned.

xil_dithermask_set_name() sets the name of the specified dither mask to the one provided.

Standard Dither Masks Provided

The XIL library creates several predefined dither masks at the time of an **xil_open**(3) call. The names of these dither masks and their suggested uses follow.

Dither Mask Name	*Suggested Use*
"dm883"	8x8x3 mask for dithering 24-bit color images to 8-bit pseudocolor images
"dm881"	8x8x1 mask for dithering 8-bit grayscale images to 1-bit images
"dm443"	4x4x3 mask for dithering 24-bit color images to 8-bit pseudocolor images
"dm441"	4x4x1 mask for dithering 8-bit grayscale images to 1-bit images

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Create and name a 2x2 single-banded dither mask:

```
XilSystemState State;
XilDitherMask dithermask;
float data[] =      { 0.0,  0.75,
                       0.25, 0.5 };

xil_dithermask_create(State, 2, 2, 1, data);
xil_dithermask_set_name(dithermask, "small_mask");
```

Perform a dither operation on a 1-banded image using "small_mask":

```
XilSystemState State;
XilDitherMask dithermask;
XilLookup cc_2color_bit;            /* 2-entry cube; black /white */
XilImage byte_image, bit_image;

dithermask = xil_dithermask_get_by_name(State, "small_mask");
xil_ordered_dither(byte_image, bit_image, cc_2color_bit, dithermask);
```

NOTES

The set of standard objects is generated for each instantiation of an *XilSystemState.* If these standard objects are deleted, they become unavailable for the duration of the current XIL session.

If you give two dither masks the same name, it is not defined which dither mask will be retrieved by a call to **xil_dithermask_get_by_name**().

SEE ALSO

xil_dithermask_create(3), **xil_dithermask_get_height**(3), **xil_open**(3).

NAME

xil_dithermask_get_height, xil_dithermask_get_width, xil_dithermask_get_nbands – read attributes of dither mask objects

SYNOPSIS

#include <xil/xil.h>

unsigned int xil_dithermask_get_height (XilDitherMask *mask***);**

unsigned int xil_dithermask_get_width (XilDitherMask *mask***);**

unsigned int xil_dithermask_get_nbands (XilDitherMask *mask***);**

DESCRIPTION

These routines control access to the dither mask object used in the **xil_ordered_dither**(3) operation. In each routine, *mask* is a handle to a dither mask.

xil_dithermask_get_width() gets the width of the specified dither mask.

xil_dithermask_get_nbands() gets the number of bands in the specified dither mask.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Get the dimensions of a dither mask:

```
XilDithermask dithermask;
unsigned int width, height, nbands;

width  = xil_dithermask_get_width (dithermask);
height = xil_dithermask_get_height (dithermask);
nbands = xil_dithermask_get_nbands (dithermask);
```

SEE ALSO

xil_dithermask_create(3), **xil_dithermask_get_by_name**(3).

NAME

xil_divide, xil_divide_by_const, xil_divide_into_const – image division operations

SYNOPSIS

#include <xil/xil.h>

void xil_divide (XilImage *src1***,**
  XilImage *src2***,**
  XilImage *dst***);**

void xil_divide_by_const (XilImage *src1***,**
  float **constants***,**
  XilImage *dst***);**

void xil_divide_into_const (float **constants***,**
  XilImage *src1***,**
  XilImage *dst***);**

DESCRIPTION

xil_divide() performs a pixel-by-pixel division of image *src2* into image *src1* and stores the result in the *dst* (destination) image.

xil_divide_by_const() performs a pixel-by-pixel division of image *constants* values into image *src1* and stores the result in the *dst* (destination) image.

xil_divide_into_const() performs a pixel-by-pixel division of image *src1* into *constants* values and stores the result in the *dst* (destination) image.

For division operations with constants and a n-band image, n float values must be provided, one per band. If the result of the operation is out of range for a particular data type, the result is clamped to the minimum or maximum value for the data type. Results for XIL_BYTE operations, for example, are clamped to 0 if they are less than 0 and 255 if they are greater than 255.

If division of a non-zero value by zero occurs, the destination value is set to the maximum value for the pixel data type. If division of zero by zero occurs, the destination value is zero. For all division cases (image into image, constant into image, image into constant), an exception is raised once for any number of occurences of division by zero.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Divide *image2* into *image1* and store the result in *dst* :

```
XilImage image1, image2, dst;

xil_divide(image1, image2, dst);
```

Divide *constants* into 4-band *image1* and store the result in *dst* :

```
XilImage image1, dst;
float constants[4];

constants[0] = 1.0;
constants[1] = 2.0;
constants[2] = 2.0;
constants[3] = 2.0;
xil_divide_by_const(image1, constants, dst);
```

Divide 4-band *image1* into *constants* and store the result in *dst* :

```
XilImage image1, dst;
float constants[4];

constants[0] = 1.0;
constants[1] = 1.0;
constants[2] = 1.0;
constants[3] = 1.0;
xil_divide_into_const(constants, image1, dst);
```

NOTES

Source and destination images must be the same data type and have the same number of bands. In-place operations are supported.

NAME

xil_edge_detection – detect edges within an image

SYNOPSIS

#include <xil/xil.h>

void xil_edge_detection (XilImage *src***,**
　　XilImage *dst***,**
　　XilEdgeDetection *edge_detection_method***);**

DESCRIPTION

This function detects edges within an image using the method specified by the *edge_detection_method* parameter. *src* is the source image handle. *dst* is the destination image handle.

edge_detection_method is an enumeration type that specifies the edge detection algorithm to be used in the operation. Currently, the only available method is XIL_EDGE_DETECT_SOBEL, which uses the following kernels:

```
     Vertical              Horizontal

  -0.5  0.0  0.5        -0.5  -1.0  -0.5
  -1.0  0.0  1.0         0.0   0.0   0.0
  -0.5  0.0  0.5         0.5   1.0   0.5
```

The XIL_EDGE_DETECT_SOBEL method performs two convolution operations on the source image, using the vertical filter to detect vertical edges and the horizontal filter to detect horizontal edges. This yields the intermediate images *a* and *b.* It then squares pixel values in *a* and *b,* yielding intermediate images *c* and *d.* To form the final destination image, it takes the square root of *c* + *d.* The convolution operations duplicate the source-image edges during the convolution, similar to using the XIL_EDGE_EXTEND edge detection method on the **xil_convolve**(3) function.

ROI Behavior

An ROI (region of interest) is used as a read mask for key pixels in the source image and as a write mask in the destination image. The edge detection operation may access data outside a source ROI as long as the key pixel remains inside.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

This example performs edge detection operation on *src* image using sobel algorithm, and writes the result into *dst.*

```
XilImage src, dst;

xil_edge_detection(src, dst, XIL_EDGE_DETECT_SOBEL);
```

NOTES

Source and destination images must be the same data type and have the same number of bands. The images need not have the same width and height. This operation cannot be performed in place.

SEE ALSO

xil_convolve(3)

NAME

xil_erode, xil_dilate – erode or dilate an image

SYNOPSIS

```
#include <xil/xil.h>

void xil_erode (XilImage src,
        XilImage dst,
        XilSel sel);

void xil_dilate (XilImage src,
        XilImage dst,
        XilSel sel);
```

DESCRIPTION

xil_erode() erodes an image.

xil_dilate() dilates an image.

src is the source image handle. *dst* is the destination image handle. *sel* is a structuring element that describes which of a source pixel's neighbors will be used as input to the operation.

ROI Behavior

An ROI (region of interest) is used as a read mask for key pixels in the source image and as a write mask in the destination image. The key pixel aligns with the output pixel and constrains which input pixels are used to generate the output. The erode and dialate operation may access data outside a source ROI as long as the key pixel remains inside.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Erode an image using a 3 x 3 "cross-shaped" structuring element with the key pixel in the center (1,1).

```
XilSystemState State;
XilImage src, dst;
XilSel sel;
unsigned int sel_data[] = { 0, 1, 0,
                            1, 1, 1,
                            0, 1, 0 };

sel=xil_sel_create (State, 3, 3, 1, 1, sel_data);

xil_erode(src, dst, sel);
```

Dilate an image using a 3 x 3 "X-shaped" structuring element with the key pixel in the upper left-hand corner (0,0).

```
XilSystemState State;
XilImage src, dst;
XilSel sel;
unsigned int sel_data[] = { 1, 0, 1,
                            0, 1, 0,
                            1, 0, 1 };

sel=xil_sel_create (State, 3, 3, 0, 0, sel_data);

xil_dilate(src, dst, sel);
```

NOTES

Source and destination images must be the same type and have the same number of bands. This operation cannot be performed in place.

SEE ALSO

xil_sel_create(3)

NAME

xil_error_diffusion – use error-diffusion dithering to convert an image into a single-band image with a colormap

SYNOPSIS

#include <xil/xil.h>

void xil_error_diffusion (XilImage *src***,**
XilImage *dst***,**
XilLookup *cmap***,**
XilKernel *distribution***);**

DESCRIPTION

This routine performs error-diffusion dithering of a *src* (source) image with a distribution matrix. It produces a single-band *dst* (destination) image. *cmap* is a lookup table with the number of output bands equal to the number of bands in the source image. *distribution* is a kernel with values between 0.0 and 1.0. This distribution matrix specifies the amount of error to distribute to the neighbors of the current pixel.

This function assumes that the entire error is distributed to the right and below the current pixel. That is, the values in the distribution kernel sum to 1.0. The only entries that can be non-zero are those to the right of and on the same row as the key entry, and those entries below the row of the key entry.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Error-diffusion dither a 3-band image into a single-band image:

```
XilImage src;                    /* 3-band source image */
XilImage dst;                    /* 1-band destination image */
XilLookup colormap;              /* colormap */
XilKernel distribution;          /* error distribution matrix */
float data[]={    0.0,        0.0,        0.0,
                  0.0,        0.0,        7.0/16.0,
                  3.0/16.0,   5.0/16.0,   1.0/16.0};

distribution = xil_kernel_create(State, 3, 3, 1, 1, data);

xil_error_diffusion(src, dst, colormap, distribution);
```

NOTES

For a discussion of error diffusion in the XIL library, consult the *XIL Programmer's Guide.*

SEE ALSO

xil_kernel_create(3), **xil_kernel_get_by_name**(3), **xil_lookup_create**(3), **xil_lookup_get_by_name**(3), **xil_kernel_get_height**(3).

NAME

xil_error_get_string, xil_error_get_id, xil_error_get_category, xil_error_get_category_string, xil_error_get_location, xil_error_get_primary, xil_error_get_object, xil_object_get_error_string, xil_object_get_type – get information about errors and the objects affected by errors

SYNOPSIS

#include <xil/xil.h>

char *xil_error_get_string (XilError *error***);**

char *xil_error_get_id (XilError *error***);**

XilErrorCategory xil_error_get_category (XilError *error***);**

char *xil_error_get_category_string (XilError *error***);**

char *xil_error_get_location (XilError *error***);**

Xil_boolean xil_error_get_primary (XilError *error***);**

XilObject xil_error_get_object (XilError *error***);**

void xil_object_get_error_string (XilObject *object,*
  char **string,*
  int *string_size***);**

XilObjectType xil_object_get_type (XilObject *object***);**

DESCRIPTION

These functions can be used by an error handler (installed with **xil_install_error_handler**(3)) to retrieve information about an error when it occurs.

xil_error_get_string () returns an error string in the currently configured language.

xil_error_get_id () returns a character string that uniquely identifies the error.

xil_error_get_category () returns the general category of the error. See **XilErrorDefines.h** for the list of categories.

xil_error_get_category_string () returns a character string that identifies the category of the error.

xil_error_get_location () returns information that indicates where the error occurred in the XIL library. By reporting this information to support personnel, you can help pinpoint the source of the problem.

xil_error_get_primary () returns TRUE if the currently reported error is the primary cause of the error. For instance, if memory runs out and an image cannot be created, then the primary error would be an XIL_ERROR_RESOURCE error at image creation. Secondary errors might also be generated as the NULL image is used internally in the XIL library.

xil_error_get_object () returns the XIL object that an error occurred on. This object can then be used in the error handler to query for additional information about the object, either through **xil_object_get_error_string** () or through direct calls to the object.

xil_object_get_error_string () creates a string with additional information about the object involved in the error. This string may then be used in the error handler to provide additional information about the error.

xil_object_get_type () returns the an enumeration constant that indicates the type of an object. This enumeration constant can be used in an error handler to take an *XilObject* and cast it to the appropriate type of *XilObject*. For example, after the object has been cast to *XilImage*, then additional information about the object is available. The following excerpt from **XilDefines.h** lists the possible *XilObjects:*

```
typedef enum {
        XIL_IMAGE,
        XIL_IMAGE_TYPE,
        XIL_LOOKUP,
        XIL_CIS,
        XIL_DITHER_MASK,
        XIL_KERNEL,
        XIL_SEL,
        XIL_ROI,
        XIL_ROI_LIST,
        XIL_HISTOGRAM,
        XIL_COLORSPACE
} XilObjectType;
```

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Create an error handler that puts out information about the category, the error, the id, and any additional object information. Also output the width of the image if the error object is an image.

```
Xil_boolean my_error_func(XilError error)
{
#define MAX 1024
        XilObject obj;
        char buffer[MAX];

        printf("XIL Error category: %s\n", xil_error_get_category_string(error));
        printf("XIL Error string: %s\n", xil_error_get_string(error));
        printf("XIL Error id: %s\n", xil_error_get_id(error));
        obj = xil_error_get_object(error);
        if (obj) {
          xil_object_get_error_string(obj,buffer,MAX);
          if (buffer[0] != 0)
            printf("XIL Object info: %s\n", buffer);
```

```
                if ( xil_object_get_type(obj) == XIL_IMAGE)
                    printf("Image Width: %d\n", xil_get_width( (XilImage)obj ));
            }
            return TRUE;


    }
```

NOTES The character pointer returned from **xil_error_get_string** () points to data internal to the error object and should not be freed or modified.

SEE ALSO **xil_default_error_handler**(3), **xil_install_error_handler**(3).

NAME

xil_export, xil_import, xil_get_exported – move an image from XIL to application space, or from application to XIL space, or determine whether an image is exported

SYNOPSIS

#include <xil/xil.h>

int xil_export (XilImage *image***);**

void xil_import (XilImage *image***,**
    Xil_boolean *change_flag***);**

int xil_get_exported (XilImage *image***);**

DESCRIPTION

xil_export () moves an image from XIL processing space into application space. This function returns *XIL_SUCCESS* if the export succeeds, and *XIL_FAILURE* if the export fails.

By calling **xil_export** () to move an image from XIL processing space into application space, the application gains access to information about how image data is stored in memory. The exported image's data must be accessed by calling **xil_get_memory_storage**(3). **xil_export** () can also be used to ensure that the image's data storage remains in main memory. This prevents the image from being moved to another device. Exporting an image may cause a reduction in the performance of operations.

xil_import () moves an image from application space into XIL processing space. An image exported for read-only purposes may be re-imported in the most efficient way if the parameter *change_flag* is set to *FALSE* (in other words, if the image was not modified). You *must* set the change flag to *TRUE* when you import an image if you make any modifications to it while it is exported.

When an application calls **xil_import** (), the XIL library is free to move the image's data to another address space and to another format; therefore, importing an image invalidates the information returned by a previous **xil_get_memory_storage**(3). If the image is exported again, the image data is unlikely to appear in the same memory location as the last time it was exported, and it's unlikely to have the same format as the last time. Therefore, **xil_get_memory_storage**(3) must be called after each **xil_export** () in order to obtain the current memory location and format for the image data.

To ensure that image data is not moved and is not reformatted, an application could export the image but never import it again. However, this prevents the XIL library from moving the image to an accelerator, if one exists, and it prevents the library from implementing its deferred execution scheme; thus, application performance is significantly degraded. After manipulating an exported image's data, it's usually best for an application to take advantage of available acceleration by importing the image; then, when it needs to manipulate data again, it can export the image and get new pointers to the data and new format information by calling **xil_get_memory_storage**(3).

xil_get_exported () returns the export status of an image. One of three possible values is returned:

0	if the image is not exported

1	if the image is exported
-1	if the image is not exportable

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

NOTES

Images created from a window with **xil_create_from_window**(3) or from a device with **xil_create_from_device**(3) cannot be exported. A description of the storage of the image cannot be requested if the image is not exported.

SEE ALSO

xil_set_memory_storage(3), **xil_get_memory_storage**(3).

NAME

xil_extrema – find maximum and minimum values of an image

SYNOPSIS

#include <xil/xil.h>

void xil_extrema (XilImage *src***,**
  float ∗*max***,**
  float ∗*min***);**

DESCRIPTION

This function finds the maximum and minimum pixel values in each bacnd of an image. *src* is the source image handle. *max* is a pointer to the floating-point array that holds the maximum value [0...nbands]. *min* is a pointer to the floating-point array that holds the minimum value [0...nbands].

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Find the maximum and minimum pixel values in a 2-banded image:

```
XilImage src;
float max[2];
float min[2];
xil_extrema(src, max, min);
```

NOTES

For an n-band image, the array of floats for *min, max* must be of size n, because each band is independently evaluated. If the maximum pointer is *NULL*, only the minimum is computed. If the minimum pointer is *NULL*, only the maximum is computed.

SEE ALSO

xil_create(3).

NAME

xil_fill – perform boundary fill from specified start point

SYNOPSIS

#include <xil/xil.h>

void xil_fill (XilImage *src***,**
    XilImage *dst***,**
    float *xseed***,**
    float *yseed***,**
    float ∗*boundary***,**
    float ∗*fill_color***);**

DESCRIPTION

This function performs a boundary fill. Given the starting coordinates, the routine fills every 4-connected pixel with the specified color until it encounters the boundary. *src* is the source image handle. *dst* is the destination image handle. *xseed* is a float that specifies the *x* start coordinate. *yseed* is a float that specifies the *y* start coordinate. *boundary* is a pointer to the floating-point array that specifies the boundary value [0...(nbands-1)] for each pixel. *fill_color* is a pointer to the floating-point array that specifies the fill color [0...(nbands-1)] for each pixel.

ROI Behavior

This function performs the fill operation on the entire source image. The filled pixels within the ROI (region of interest) are output to the destination image.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

For this example, the source and destination images contain 2 bands. Perform boundary fill starting at (x,y) = (7,3).

```
XilImage src;
XilImage dst;
float xseed = 7.0;
float yseed = 3.0;
float boundary[2] = {255.0, 0.0};
float fill_color [2] =  {0.0,255.0};
xil_fill(src, dst, xseed, yseed, boundary, fill_color);
```

NOTES

Source and destination images must be the same data type, and have the same number of bands. For an n-band image, the array of floats for *boundary* and *fill_color* must be of size n. A pixel that matches each band in the specified *boundary* value is a boundary pixel. Only pixels that are changed to the fill color are output to the destination image. In-place operations are supported.

SEE ALSO

xil_create(3), **xil_roi_create**(3).

NAME

xil_get_attribute, xil_set_attribute – get and set the client attributes of images

SYNOPSIS

#include <xil/xil.h>

int xil_get_attribute (XilImage *image***,**
    char **attribute***,**
    void ***value***);**

int xil_set_attribute (XilImage *image***,**
    char **attribute***,**
    void **value***);**

DESCRIPTION

These routines get and set values of client attributes of images. Names of the attributes can be arbitrarily assigned and are simply saved for later retrieval. *attribute* is the name of the attribute whose value is to be retrieved or set. *value* is the status of the specified attribute.

xil_get_attribute() returns XIL_SUCCESS if the attribute is available, and XIL_FAILURE if the specified attribute is not available.

xil_set_attribute() returns XIL_SUCCESS if the attribute is successfully set, and XIL_FAILURE otherwise.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Set the date that a photograph was taken:

```
XilImage image;
char *attribute;

status = xil_set_attribute (image, "DATE", (void *)date);
if(status==XIL_FAILURE)
  fprintf(stderr,"Failed to set DATE attribute0);
```

Get the favorite ice cream flavor of the person in the photograph:

```
XilImage image;
char *attribute;

status = xil_get_attribute (image, "favorite flavor", (void **)&(flavor));
if(status==XIL_FAILURE)
  fprintf(stderr,"Failed to get flavor attribute0);
```

NOTES

These functions are not intended to to be used as a database interface. If the image does not contain the specified attribute, the parent is searched for the attribute, then the parent's parent is searched, and so on, until there are no more parents.

SEE ALSO

xil_get_device_attribute(3), **xil_set_device_attribute**(3), **xil_cis_get_attribute**(3), **xil_cis_set_attribute**(3).

NAME

xil_get_by_name, xil_get_name, xil_set_name – get and set an image object name and get a handle to an image by specifying a name

SYNOPSIS

#include <xil/xil.h>

XilImage xil_get_by_name (XilSystemState *State***,**
    char ∗*name***);**

char∗ **xil_get_name (XilImage** *image***);**

void xil_set_name (XilImage *image***,**
    char ∗*name***);**

DESCRIPTION

Use these functions to assign names to image objects, to read an image's name, and to retrieve image objects by name.

xil_get_by_name() returns the handle to the image with the specified name *name.* If such an image does not exist, NULL is returned. **xil_get_by_name** () does not make a copy of the image.

xil_get_name() returns a copy of the specified image's name. A call to **free** (3) should be used to free the space allocated by **xil_get_name**()
If the specified image has no name, NULL is returned.

xil_set_name() sets the name of the specified image to the one provided.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Create a 5x5 3-band blank test image called "empty5x5x3":

```
XilSystemState State;
XilImage image;
float values[] = { 0.0, 0.0, 0.0 };

image = xil_create(State,5,5,3,XIL_BYTE);
xil_set_value(image, values);
xil_set_name(image, "empty5x5x3");
```

Use an image named "empty5x5x3" to zero a portion of another image:

```
XilSystemState State;
XilImage zero_image, src, src_child;

zero_image = xil_get_by_name (State,"empty5x5x3");
src_child = xil_create_child (src, 100, 100, 5, 5, 1, 3);
xil_multiply (src_child, zero_image, src_child);
```

NOTES

If you give two images the same name, it is not defined which image will be retrieved by a call to **xil_get_by_name**().

SEE ALSO

xil_create_child(3).

NAME

xil_get_child_offsets – get values of the offsets into a parent image

SYNOPSIS

#include <xil/xil.h>

void xil_get_child_offsets (XilImage *image***,**
  unsigned int **offsetX***,**
  unsigned int **offsetY***,**
  unsigned int **offsetBand***);**

DESCRIPTION

This function gets the values of the offsets into a parent image that were used in the **xil_create_child**(3) call that created the specified child image. *offsetX* is the horizontal offset in pixels from the upper-left corner of the parent image to the upper-left corner of the child image. *offsetY* is the vertical offset in pixels from the upper-left corner of the parent image to the upper-left corner of the child image. *offsetBand* is the offset in bands, starting from the first band of the parent image to the first band in the child image.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Get the offsets used to create a child image:

```
XilImage image;
unsigned int x_offset, y_offset, band_offset;

xil_get_child_offsets(image, &x_offset, &y_offset, &band_offset);
```

SEE ALSO

xil_create_child(3).

NAME

xil_get_datatype – get an image's data type

SYNOPSIS

#include <xil/xil.h>

XilDataType xil_get_datatype (XilImage *image***);**

DESCRIPTION

This function gets the data type of an *image*. The possible types returned are XIL_BIT, XIL_BYTE, and XIL_SHORT. This function may be called on all images.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Get the datatype of an image:

```
XilImage image;
XilDataType datatype;

datatype = xil_get_datatype (image);
```

SEE ALSO

xil_get_imagetype(3), **xil_get_info**(3), **xil_get_width**(3), **xil_get_height**(3), **xil_get_nbands**(3), **xil_get_size**(3).

NAME

xil_get_device_attribute, xil_set_device_attribute – get and set the values of attributes of device images

SYNOPSIS

#include <xil/xil.h>

int xil_get_device_attribute (XilImage *image***,**
    char **attribute***,**
    void ***value***);**

int xil_set_device_attribute (XilImage *image***,**
    char **attribute***,**
    void **value***);**

DESCRIPTION

These routines get and set the values of attributes of device images. *image* is a handle to a device image. *attribute* is the name of an attribute, and *value* is the attribute's value. Attribute names and their possible values are defined by the group that writes the device handler.

xil_get_device_attribute() gets a device-specific *attribute.* It returns XIL_SUCCESS if the attribute is available, and XIL_FAILURE if the specified attribute is not available.

xil_set_device_attribute() sets a device-specific *attribute.* It returns XIL_SUCCESS if the attribute is successfully set, and XIL_FAILURE otherwise.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Set the brightness of a frame-grabber input image:

```
int brightness;
brightness = 100;
XilImage framegrabber_image;

status = xil_set_device_attribute(framegrabber_image, "BRIGHTNESS",
        (void *)brightness);
if(status==XIL_FAILURE)
  fprintf(stderr,"Setting BRIGHTNESS attribute failed0);
```

Get the contrast of a frame-grabber input image:

```
int contrast;
XilImage framegrabber_image;

status = xil_get_device_attribute(framegrabber_image, "CONTRAST",
        (void **)&contrast);
if(status==XIL_FAILURE)
  fprintf(stderr,"Getting CONTRAST attribute failed0);
```

NOTES

xil_set_device_attribute() is used to set the attributes of an existing device image; it cannot be used to initialize attribute values before creating the device image. To initialize device attributes, use **xil_device_set_value**().

SEE ALSO

xil_create_from_window(3), **xil_create_from_device**(3), **xil_get_attribute**(3), **xil_get_readable**(3), **xil_get_writable**(3), **xil_device_create**(3), **xil_device_set_value**(3).

NAME

xil_get_imagetype – get an XilImageType object

SYNOPSIS

#include <xil/xil.h>

XilImageType xil_get_imagetype (XilImage *image***);**

DESCRIPTION

This function returns an *XilImageType* object that contains information about the size, data type, and color space of an *image.* This function may be called on all images. An *XilImageType* object describes the characteristics of an image that will be generated (or expected) by a particular device (for example, a frame grabber or an output device). The characteristics of an *XilImageType* object are *xsize, ysize, nbands, datatype,* and *colorspace.* You obtain an *XilImageType* object from a call to **xil_cis_get_output_type**(3) or **xil_cis_get_input_type**(3). You use an *XilImageType* object to create images (via an **xil_create_from_type**(3) call) that will be compatible with a given device, compressor, and so on.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Get the imagetype of a particular image:

```
XilImage image;
XilImageType imagetype;

imagetype = xil_get_imagetype (image);
```

NOTES

After *image* is destroyed, the handle to the *XilImageType* object is no longer valid.

SEE ALSO

xil_create_from_type(3), **xil_cis_get_output_type**(3), **xil_cis_get_input_type**(3).

NAME

xil_get_info – get information about the parameters of an image

SYNOPSIS

#include <xil/xil.h>

void xil_get_info (XilImage *image***,**
    unsigned int ∗*width***,**
    unsigned int ∗*height***,**
    unsigned int ∗*nbands***,**
    XilDataType ∗*datatype***);**

DESCRIPTION

This function gets the following *image* parameters: *width*, *height*, *nbands* (number of bands), and *datatype.* This function may be called on all images. Use **xil_get_imagetype**(3) to get a handle to an object with the same characteristics as a given image; this handle can then be used in **xil_create_from_type**(3) calls.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Get all the parameters that describe a particular image:

```
XilImage image;
unsigned int width, height, nbands;
XilDataType datatype;

xil_get_info (image, &width, &height, &nbands, &datatype);
```

SEE ALSO

xil_get_datatype(3), **xil_get_imagetype**(3), **xil_get_width**(3), **xil_get_height**(3), **xil_get_nbands**(3), **xil_get_size**(3), **xil_create_from_type**(3).

NAME

xil_get_memory_storage, xil_set_memory_storage – get and set memory storage

SYNOPSIS

#include <xil/xil.h>

Xil_boolean xil_get_memory_storage (XilImage *image,*
  XilMemoryStorage **storage***);**

void xil_set_memory_storage (XilImage *image,*
  XilMemoryStorage **storage***);**

DESCRIPTION

Use these functions when you want to get or set the data in an image.

xil_get_memory_storage () returns a description of how an exported image is stored in system memory. Storage for this description must be allocated by the user. **xil_get_memory_storage** () returns TRUE if the memory storage could be obtained, and FALSE otherwise. This can be used before calls such as **fread**(3S) to test whether the data is available for the desired operation.

The information returned by **xil_get_memory_storage** () is valid only while the image remains exported. After the image is imported, both the address at which the image's pixel values are located and the pixel layout in memory is likely to change. Thus, the information that was returned by **xil_get_memory_storage** () prior to the import is no longer valid. Trying to access pixel values using invalid pointers to the data or using invalid information about the pixel layout can cause serious problems in an application.

In the XIL library, multibanded images - except for 1-bit images - are stored in a pixel-sequential format. The following attributes are only exposed to the application if the image is exported:

Distance to the same pixel on the next horizontal scanline (the vertical stride)
Distance to the next pixel on the same scanline (the pixel stride)
Starting address of the image

For 1-bit multibanded images, the data is stored in a band-sequential manner. The export of 1-bit images exposes four private attributes that define the image storage:

Distance in bytes to the byte of the same pixel in the next scanline
Distance in bytes to the same pixel of the next band
Number of bits to offset to the first pixel
Byte starting address of the image data

User data may be imported after image creation if it meets the layout and data type criteria described.

XilMemoryStorage is defined as follows:

```
typedef union XilMemoryStorageBit {
    struct {
            Xil_unsigned8* data;              /* pointer to first byte of image */
            unsigned short scanline_stride;   /* the number of bytes between scanlines */
            unsigned long band_stride;        /* the number of bytes between bands */
            unsigned char offset;             /* the number of bits to the first pixel */
    } bit;

    struct XilMemoryStorageByte {
            Xil_unsigned8* data;              /* pointer to the first byte of the image */
            unsigned long scanline_stride;    /* the number of bytes between scanlines */
            unsigned short pixel_stride;      /* the number of bytes between pixels */
    } byte;

    struct XilMemoryStorageShort {
            Xil_signed16* data;               /* pointer to the first word of the image */
            unsigned long scanline_stride;    /* the number of 16 bit words between scanlines */
            unsigned short pixel_stride;      /* the number of 16 bit words between pixels */
    } shrt;
}XilMemoryStorage;
```

When manipulating the data, it's important to use the *scanline_stride* and *pixel_stride* information returned by **xil_get_memory_storage** (); you cannot make assumptions about the image's format in memory storage. For example, some accelerators may not handle 3-banded RGB images while they do handle 4-banded (RGBA) images. For these accelerators, the memory storage code converts 3-banded images into 4-banded images when the first accelerator function is called on the image data. If the image is then exported, the XIL library returns a 3-banded child of a 4-banded image as the data layout for the 3-banded image that was imported. This means that the code written on the exported data cannot assume a 3-pixel layout and cannot skip to the beginning of the next pixel by simply doing a **src++*.

xil_set_memory_storage () allows an application to specify the memory used for an *image*. This *storage* is specified with the same *XilMemoryStorage* structure that **xil_get_memory_storage** () uses. The memory must be both readable and writable. After **xil_set_memory_storage** () has been called, the image resides in the specified memory *only* while the image remains exported.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Fill an image with the contents of a file. Note that you must export the image before you can call **xil_get_memory_storage**(). Likewise, you must import it when you are done using the data.

```
XilImage image;
int width, height, nbands;
XilDataType datatype;
XilMemoryStorage storage;
Xil_boolean status;
char *infile = "input_image";

xil_export(image);
status = xil_get_memory_storage(image, &storage);
if(status == FALSE) {
  /* XIL's error handler will print an error msg to stderr */
  exit(1);
}
int h, w, n;
Xil_unsigned8* scanline = storage.byte.data;
xil_get_info(image, &width, &height, &nbands, &datatype);
/*
 * The following loop uses fread to read from an infile. The same
 * loop could be used to write to an outfile by replacing fread with
 * fwrite and replacing the infile with an outfile
 */
for(h=0; h<height; h++) {
   Xil_unsigned8* row = scanline;
   for(w=0; w<width; w++) {
      fread((char*)row, nbands, sizeof(Xil_unsigned8), infile);
      row += storage.byte.pixel_stride;
   }
   scanline += storage.byte.scanline_stride;
}
xil_import(image);
```

NOTES

The information returned from **xil_get_memory_storage** () or set by **xil_set_memory_storage** () is valid only as long as the image is exported. Memory resources allocated by the XIL library are freed by the XIL library. Memory resources allocated by an application are not freed by the XIL library.

SEE ALSO

xil_import(3), **xil_export**(3).

NAME

xil_get_origin, xil_get_origin_x, xil_get_origin_y, xil_set_origin – get and set the origin of an image

SYNOPSIS

#include <xil/xil.h>

void xil_get_origin (XilImage *image***,**
    float **x***,**
    float **y***);**

float xil_get_origin_x (XilImage *image***);**

float xil_get_origin_y (XilImage *image***);**

void xil_set_origin (XilImage *image***,**
    float *x***,**
    float *y***);**

DESCRIPTION

These functions get and set the conceptual origin of an image. In the XIL library, each image has a pair of floating-point numbers that represents a conceptual origin. The default origin for an image when it is created is the upper left corner of the image (0.0, 0.0). When an operation is performed, the origins of the source and destination images are aligned. The floating-point origin values are rounded to integers for this purpose.

For all nongeometric operators, the following semantics are used to determine the extent of the processing. The source image or images and the destination image are conceptually moved so that their origins are coincident. The intersection of the source and destination images then forms the destination bounds. Only the area of intersection is modified in the destination image, and only the area of intersection in the source is used by the operator. This is very similar to the way in which regions of interest (ROIs) are handled.

Geometric operations behave a little differently, in that after the source and destination origins have been lined up, the bounds of the source image are geometrically transformed and then interesected with the bounds of the destination image. Note that as a result of the transform, the intersection may result in a nonrectangular region in the destination where modification can occur. ROIs are also handled in the same manner.

If the semantic described above does not produce any overlap, no pixels in the destination are touched.

xil_get_origin() gets the *x* and *y* coordinates of the origin of an *image.*

xil_get_origin_x() gets the *x* coordinate of the origin of an *image.*

xil_get_origin_y() gets the *y* coordinate of the origin of an *image.*

xil_set_origin() sets the *x* and *y* coordinates of the origin of an *image.*

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Move the origin of an image +20.0 in *x* and -30.0 in *y* :

```
XilImage image;
float x, y;

xil_get_origin (image, &x, &y);
x += 20.0;
y -= 30.0;
xil_set_origin (image, x, y);
```

NOTES

The origin is not constrained to lie within the boundaries of the image.

NAME

xil_get_parent – get a parent image

SYNOPSIS

#include <xil/xil.h>

XilImage xil_get_parent (XilImage *image***);**

DESCRIPTION

This function returns a handle to the parent of a child *image.* If the image is not a child image, then *NULL* is returned.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Get the parent of a child image:

```
XilImage base_image, child_image;

base_image = xil_get_parent (child_image);
```

NOTES

Child images are not hierarchical; the parent of a child image must always be a parent image.

SEE ALSO

xil_create_child(3)

NAME

xil_get_readable, xil_get_writable – return TRUE if an image can be used as a source or destination

SYNOPSIS

#include <xil/xil.h>

Xil_boolean xil_get_readable (XilImage *image***);**

Xil_boolean xil_get_writable (XilImage *image***);**

DESCRIPTION

xil_get_readable() returns TRUE if an *image* can be used as a source. Some device images cannot be used as source images.

xil_get_writable() returns TRUE if an *image* can be used as a destination. Some device images cannot be used as destination images.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Determine whether a particular image can be used as a source:

```
XilImage image, src;
Xil_boolean isreadable;

isreadable = xil_get_readable(image);
if(isreadable)
  src = image;
```

Determine whether a particular image can be used as a destination:

```
XilImage image, dst;
Xil_boolean iswritable;

iswritable = xil_get_writable(dst);
if(iswritable)
  dst = image;
```

SEE ALSO

xil_create_from_device(3), **xil_create_from_window**(3).

NAME

xil_get_roi, xil_set_roi – get or set an image's ROI

SYNOPSIS

#include <xil/xil.h>

XilRoi xil_get_roi (XilImage *image***);**

void xil_set_roi (XilImage *image***,**
  XilRoi *roi***);**

DESCRIPTION

These functions get and set the region of interest (ROI) associated with an image.

xil_get_roi() returns a copy of the ROI associated with the specified *image.*

xil_set_roi() sets the ROI associated with the specified *image* to the one supplied.

ROI Behavior

An efficient way to specify an ROI that encompasses an entire image is to set the image's ROI to NULL.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Get the ROI associated with an image, remove a rectangular region from the ROI, and replace the image's ROI with the modified one. Then destroy the ROI:

```
XilSystemState State;
XilImage image;
XilRoi roi;

roi = xil_get_roi (image);
if (roi == NULL) {
   /* The image had no ROI associated with it,
     create one that encompasses the whole image */
   roi = xil_roi_create (State);
   xil_roi_add_rect (roi, 0, 0, xil_get_width(image), xil_get_height(image));
}
xil_roi_subtract_rect (roi, 10, 10, 20, 20);
xil_set_roi (image, roi);
xil_roi_destroy (roi);
```

SEE ALSO

xil_roi_add_rect(3), **xil_roi_create**(3), **xil_roi_create_copy**(3), **xil_roi_destroy**(3), **xil_roi_intersect**(3), **xil_roi_translate**(3) **xil_roi_add_image**(3), **xil_roi_add_region**(3), **xil_roi_get_as_image**(3), **xil_roi_get_as_region**(3), **xil_roi_subtract_rect**(3), **xil_roi_unite**(3).

NAME

xil_get_width, xil_get_height, xil_get_nbands, xil_get_size – get width, height, number of bands, or size of image

SYNOPSIS

#include <xil/xil.h>

unsigned int xil_get_width (XilImage *image***);**

unsigned int xil_get_height (XilImage *image***);**

unsigned int xil_get_nbands (XilImage *image***);**

void xil_get_size (XilImage *image***,**
    unsigned int ∗*width***,**
    unsigned int ∗*height***);**

DESCRIPTION

xil_get_width() gets the width of an *image.*

xil_get_height() gets the height of an *image.*

xil_get_nbands() gets the number of bands in an *image.*

xil_get_size() gets the *width* and *height* of an *image.*

These functions may be called on all images.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Get the width and height of an image:

```
unsigned int width, height;
XilImage image;

width = xil_get_width (image);
height = xil_get_height (image);
```

Or alternatively:

```
xil_get_size (image, *width, *height);
```

SEE ALSO

xil_get_imagetype(3), **xil_get_datatype**(3), **xil_get_info**(3).

NAME

xil_histogram – generate histogram data from an image

SYNOPSIS

#include <xil/xil.h>

void xil_histogram (XilImage *src***,**
  XilHistogram *histogram***,**
  unsigned int *skip_x***,**
  unsigned int *skip_y***);**

DESCRIPTION

This routine accumulates histogram information from the source image into a histogram object that was created with the **xil_histogram_create**( ) function.

src is the source image handle. *histogram* is the handle for the histogram object that holds the histogram data. *skip_x* and *skip_y* indicate the frequency with which pixels will be counted. If *skip_x* is set to 1, **xil_histogram**() counts every pixel on a scanline; if it is set to 2, the function counts every other pixel; and so on. The value of *skip_y* has an analogous effect on how **xil_histogram**() counts pixels in the vertical direction. Using values greater than 1 allows a faster construction of a histogram by considering fewer pixels.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Generate the histogram of an image, only counting every third pixel:

```
XilImage src;
XilHistogram histogram;

xil_histogram(src, histogram, 3, 3);
```

NOTES

The number of bands in the histogram must match the number of bands in the source image. The data values in the histogram are not initialized to zero at the beginning of this operation, thereby allowing the generation of multi-image histograms.

SEE ALSO

xil_histogram_create(3)

NAME

xil_histogram_create, xil_histogram_destroy – create or destroy histogram

SYNOPSIS

#include <xil/xil.h>

XilHistogram xil_histogram_create (XilSystemState *State***,**
    unsigned int *nbands***,**
    unsigned int **nbins***,**
    float **low_value***,**
    float **high_value***);**

void xil_histogram_destroy (XilHistogram *histogram***);**

DESCRIPTION

These routines create and destroy histogram objects. Histograms are used to accumulate level information from images. XIL histograms can have arbitrary numbers of bands, but the number of bands must match the number of bands in the image that is to be histogrammed. A 3-banded histogram, for example, contains a cube of information that reflects pixel values in the three bands independently.

State is the XIL system state. *nbands* is the number of independent bands in the histogram. *nbins* is a pointer to an array that contains the number of bins for each band. These bins are used to hold information about gray or color levels.

low_value is a pointer to an array of floats that defines the value of the first bin for each band, and *high_value* is a pointer to an array of floats that defines the value of the last bin for each band. For each of the arrays *nbins*, *low_value*, and *high_value*, the number of elements in the array must match the number of bands in the image.

xil_histogram_destroy( ) destroys the specified histogram object.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Create a histogram structure appropriate for calculating the histogram of a 3-band XIL_BYTE image:

```
XilSystemState State;
XilHistogram histogram;
unsigned int nbins[3] = {32,32,32};
float low_value[3] = {0.0, 0.0, 0.0};
float high_value[3] = {255.0, 255.0, 255.0};

histogram = xil_histogram_create (State, 3, nbins, low_value, high_value);
```

SEE ALSO

xil_histogram(3), **xil_histogram_get_nbands**(3), **xil_histogram_get_nbins**(3), **xil_histogram_get_values**(3), **xil_histogram_get_info**(3), **xil_choose_colormap**(3).

NAME

xil_histogram_get_by_name, xil_histogram_get_name, xil_histogram_set_name – get and set a histogram object name and get a handle to a histogram by specifying a name

SYNOPSIS

#include <xil/xil.h>

XilHistogram xil_histogram_get_by_name (XilSystemState *State***,**
char ∗*name***);**

char∗ **xil_histogram_get_name (XilHistogram** *histogram***);**

void xil_histogram_set_name (XilHistogram *histogram***,**
char ∗*name***);**

DESCRIPTION

Use these functions to assign names to histogram objects, set a histogram's name, and to retrieve histogram objects by name.

xil_histogram_get_by_name() returns the handle to the histogram with the specified name *name.* If such a histogram does not exist, NULL is returned. **xil_histogram_get_by_name**() does not make a copy of the histogram.

xil_histogram_get_name() returns a copy of the specified *histogram's* name. A call to **free** (3) should be used to free the space allocated by **xil_histogram_get_name**(). If the specified histogram has no name, NULL is returned.

xil_histogram_set_name() sets the *name* of the specified *histogram* to the one provided.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Create and name a histogram from a single-band byte reference image:

```
XilSystemState State;
XilImage ref_image;
XilHistogram histogram;

histogram = xil_histogram_create(State, 1, 256, 0.0, 255.0);
xil_histogram(ref_image, histogram, 1, 1);
xil_histogram_set_name(histogram, "reference");
```

Get a histogram named "reference" for comparison:

```
XilSystemState State;
XilHistogram histogram;

histogram = xil_histogram_get_by_name(State, "reference");
```

NOTES

If you give two histograms the same name, it is not defined which histogram will be retrieved by a call to **xil_get_by_name**().

SEE ALSO **xil_histogram_create**(3), **xil_histogram**(3).

NAME

xil_histogram_get_nbands, xil_histogram_get_nbins, xil_histogram_get_limits, xil_histogram_get_values, xil_histogram_get_info – histogram attributes

SYNOPSIS

#include <xil/xil.h>

unsigned int xil_histogram_get_nbands (XilHistogram *histogram***);**

void xil_histogram_get_nbins (XilHistogram *histogram***,**
    unsigned int ∗*nbins***);**

void xil_histogram_get_limits (XilHistogram *histogram***,**
    float ∗*low_value,*
    float ∗*high_value***);**

void xil_histogram_get_values (XilHistogram *histogram***,**
    unsigned int ∗*data***);**

void xil_histogram_get_info (XilHistogram *histogram***,**
    unsigned int ∗*nbands,*
    unsigned int ∗*nbins,*
    float ∗*low_value,*
    float ∗*high_value***);**

DESCRIPTION

These routines read the values of histogram attributes and the intensity-level information stored in the histograms. Histograms are used to obtain information about the distribution of pixel values in an image. Create histograms with **xil_histogram_create**(3).

xil_histogram_get_nbands() returns the number of bands represented by the *histogram.* For example, a histogram with three bands can be thought of as a cube of data, with each axis representing a single band.

xil_histogram_get_nbins() fills in a user-supplied array, *nbins,* with values representing the number of *histogram* bins for each histogram band.

xil_histogram_get_limits() fills in user-supplied arrays, *low_value* and *high_value,* with floating point numbers that represent the value of the first bin and last bin in each band.

xil_histogram_get_values() fills in the user-supplied array, *data,* with the unsigned integer values that make up the histogram data. The data are aligned so that values along the first band's axis are contiguous.

xil_histogram_get_info() combines the function of other attribute functions. *nbands* is filled with the number of bands in the histogram; *nbins* is filled with the number of bins per band, one for each band. *low_value* and *high_value* are arrays that contain the low and high values for each band.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Create an array to hold the histogram data and retrieve the data:

```
XilHistogram histogram;
unsigned int nbands;
unsigned int *bins, *data;
int i, total_entries;

nbands = xil_histogram_get_nbands (histogram);
bins = (unsigned int*) malloc(nbands * sizeof(unsigned int));
xil_histogram_get_nbins (histogram, bins);
total_entries = 1;
for (i=0; i<nbands; i++)
        total_entries *= bins[i];
data = (unsigned int*) malloc(total_entries * sizeof(unsigned int));
xil_histogram_get_values(histogram, data);
```

SEE ALSO

xil_histogram(3), **xil_histogram_create**(3), **xil_histogram_destroy**(3).

NAME

xil_imagetype_get_by_name, xil_imagetype_get_name, xil_imagetype_set_name – get and set an imagei-type object name and get a handle to an image type by specifying its name

SYNOPSIS

#include <xil/xil.h>

XilImageType xil_imagetype_get_by_name (XilSystemState *State***,**
    char **name***);**

char* xil_imagetype_get_name (XilImageType *imagetype***);**

void xil_imagetype_set_name (XilImageType *imagetype***,**
    char **name***);**

DESCRIPTION

Use these functions to assign names to image type objects, to read the names of image types, and to retrieve image type objects by name.

xil_imagetype_get_by_name() returns the handle to the image type object with the specified name *name.* If such an image type object does not exist, NULL is returned. **xil_get_by_name**() does not make a copy of the image type object.

xil_imagetype_get_name() returns a copy of the specified image type object's name. A call to **free** (3) should be used to free the space allocated by **xil_imagetype_get_name**(). If the specified image type object has no name, NULL is returned.

xil_imagetype_set_name() sets the name of the specified image type object to the one provided.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Create an image type object that characterizes a particular display and call it "Sun_bw2_hires":

```
XilSystemState State;
XilImage image;
XilImageType imagetype;
unsigned int height, width, nbands;

width = 1600;
height = 1280;
nbands = 1;
image = xil_create(State, width, height, nbands, XIL_BIT);
imagetype = xil_get_imagetype(image);
xil_imagetype_set_name(imagetype, "Sun_bw2_hires");
```

Use an image type object named "Sun_bw2_hires" to create an image appropriate for display on a particular frame buffer:

```
XilSystemState State;
XilImageType imagetype;
XilImage display_image;

imagetype = xil_imagetype_get_by_name(State,"Sun_bw2_hires");
display_image = xil_create_from_type(State, imagetype);
```

NOTES

If you give two image type objects the same name, it is not defined which image type object will be retrieved by a call to **xil_imagetype_get_by_name**().

NAME

xil_imagetype_get_datatype – get data type of an image type object

SYNOPSIS

#include <xil/xil.h>

XilDataType xil_imagetype_get_datatype (XilImageType *imagetype***);**

DESCRIPTION

xil_imagetype_get_datatype() gets the data type of an image type object. XilImageType is an enumerated type. Its possible values are XIL_BIT, XIL_BYTE, and XIL_SHORT.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Get the data type of an image type object:

```
XilImageType imagetype;
XilDataType datatype;

datatype = xil_imagetype_get_datatype (imagetype);
```

SEE ALSO

xil_get_imagetype(3), **xil_imagetype_get_info**(3), **xil_imagetype_get_width**(3), **xil_imagetype_get_height**(3), **xil_imagetype_get_nbands**(3), **xil_imagetype_get_size**(3).

NAME

xil_imagetype_get_info – get information about the parameters of an image type object

SYNOPSIS

#include <xil/xil.h>

void xil_imagetype_get_info (XilImageType *imagetype***,**
        unsigned int ∗*width***,**
        unsigned int ∗*height***,**
        unsigned int ∗*nbands***,**
        XilDataType ∗*datatype***);**

DESCRIPTION

xil_imagetype_get_info() gets the following image type object parameters: width, height, nbands (number of bands), and datatype.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Get all the parameters that describe a particular image type object:

```
XilImageType imagetype;
unsigned int width, height, nbands;
XilDataType datatype;

xil_imagetype_get_info (imagetype, &width, &height, &nbands, &datatype);
```

SEE ALSO

xil_get_imagetype(3), **xil_imagetype_get_datatype**(3), **xil_imagetype_get_width**(3), **xil_imagetype_get_height**(3), **xil_imagetype_get_nbands**(3), **xil_imagetype_get_size**(3).

NAME

xil_imagetype_get_width, xil_imagetype_get_height, xil_imagetype_get_nbands, xil_imagetype_get_size – get width, height, number of bands, or size of image type objects

SYNOPSIS

#include <xil/xil.h>

unsigned int xil_imagetype_get_width (XilImageType *imagetype***);**

unsigned int xil_imagetype_get_height (XilImageType *imagetype***);**

unsigned int xil_imagetype_get_nbands (XilImageType *imagetype***);**

void xil_imagetype_get_size (XilImageType *imagetype***,**
　　unsigned int **width***,**
　　unsigned int **height***);**

DESCRIPTION

xil_imagetype_get_width() returns the width of an image type object.

xil_imagetype_get_height() returns the height of an image type object.

xil_imagetype_get_nbands() returns the number of bands in an image type object.

xil_imagetype_get_size() returns the size of an image type object, returning its width and height.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Get the width and height of an image type object:

```
unsigned int width, height;
XilImageType imagetype;

width = xil_imagetype_get_width (imagetype);
height = xil_imagetype_get_height (imagetype);
```

Or alternatively:

```
xil_imagetype_get_size (imagetype, &width, &height);
```

SEE ALSO

xil_get_imagetype(3), **xil_imagetype_get_datatype**(3), **xil_imagetype_get_info**(3).

NAME

xil_install_error_handler, xil_default_error_handler, xil_remove_error_handler, xil_call_next_error_handler – install or remove a customized error handler, or use the default version

SYNOPSIS

#include <xil/xil.h>

int xil_install_error_handler (XilSystemState *State*,
    XilErrorFunc *func***);**

void xil_remove_error_handler (XilSystemState *State*,
    XilErrorFunc *func***);**

Xil_boolean xil_call_next_error_handler (XilError *error***);**

Xil_boolean xil_default_error_handler (XilError *error***);**

DESCRIPTION

Errors and warnings in the XIL library are dispatched through an error handling routine. Users can provide their own customized error function or use the XIL default routine. Users can also chain error handlers to allow individual error handlers to handle only a certain type of error.

xil_install_error_handler() installs a user-provided customized error function. Inside this function, calls can be made to the various **xil_error_get_*** routines to get information about the error. The return value from this error handler can be used by any error handlers further up the chain to determine whether the error has been successfully handled. The most recently installed error handler is called first, then the next most recently installed error handler, and so on, so that the last error handler to be installed is the first to be called.

xil_remove_error_handler() removes an error function from the error handler chain. It can be used to remove the default error handler from the error handler chain.

xil_call_next_error_handler() can be called from within an error handler to allow an error handler further down the chain to handle the error.

xil_default_error_handler() prints an informative message about errors and warnings to the standard error output. The default error handler always returns *TRUE* and is always the last error handler on the error handler chain.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

```
/* Print the standard error message.
 * If the error is a RESOURCE error, then quit.
 */
Xil_boolean resource_errors(XilError error)
{
        int ret_val;
        ret_val = xil_call_next_error_handler(error);
        if (xil_error_get_category(error) == XIL_ERROR_RESOURCE)
          exit(1);
        return ret_val;
}

main()
{
        XilSystemState State;

        State=xil_open();
        if (State==NULL) {
          printf("Couldn't initialize XIL\n");
          exit(1);
        }
        xil_install_error_handler(State,resource_errors);
}
```

NOTES

Only certain XIL functions can be called from within an error handler. For more information, see the *XIL Programmer's Guide.*

SEE ALSO

xil_error_get_string(3), **xil_default_error_handler**(3).

NAME

xil_interpolation_table_create, xil_interpolation_table_destroy – create or destroy an interpolation table object

SYNOPSIS

#include <xil/xil.h>

XilInterpolationTable xil_interpolation_table_create (XilSystemState *state***,**
unsigned int *kernel_size***,**
unsigned int *subsamples***,**
float **data***);**

void xil_interpolation_table_destroy (XilInterpolationTable *table***);**

DESCRIPTION

These routines create and destroy interpolation table objects. An XilInterpolationTable object is an array of 1xn kernels which represents the interpolation filter in either the horizontal or vertical direction. The datatype of the table is XIL_FLOAT.

The parameter *state* is the XIL system state, *kernel_size* is the size of the kernel, *subsamples* is the number of subsamples between pixels, and *data* is the data of the interpolation table. There is no limit or restrictions on the kernel size or the number of subsamples.

Each subsample requires a separate set of kernel data. Thus, *n* subsamples require *n* * *kernel_size* data elements. For example, a horizontal interpolation table with a kernel size of 7 elements and a pixel subsampling of 3 requires 21 data elements; the first subsample uses the first 7 data elements, the second subsample uses the next 7 data elements, and the third subsample uses the last 7 data elements. If both the horizontal and vertical interpolation tables are NULL, nearest neighbor interpolation is performed.

xil_interpolation_table_destroy () destroys the specified interpolation table object.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Create a horizontal interpolation table with seven kernel elements and two subsamples between pixels:

```
XilSystemState state;
XilInterpolationTable horiz_table;
float *data;

horiz_table = xil_interpolation_table_create (state, 7, 2, data);
```

NOTES

The key element in a kernel is the center element; for even-sized kernels, the key elements is the first of the two center elements. Thus, for an 8-element kernel, the key value is the fourth element, which has the array index 3. The key element's array index can be computed as an integer calculation:

```
int array_index = (kernel_size - 1) / 2
```

To preserve the source image's intensity in the destination, an individual kernel's values should sum to one.

SEE ALSO

xil_interpolation_table_get_subsamples(3), **xil_interpolation_table_get_kernel_size**(3), **xil_interpolation_table_get_data**(3), **xil_state_get_interpolation_tables**(3).

NAME

xil_interpolation_table_get_data – get the data of an interpolation table object

SYNOPSIS

#include <xil/xil.h>

float * xil_interpolation_table_get_data (XilInterpolationTable *table***);**

DESCRIPTION

xil_interpolation_table_get_data () gets the *data* from an interpolation table object *table*. Enough memory should be allocated to hold the *subsamples * kernel_size* floating point data elements.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide*.

EXAMPLES

Get data of an interpolation table object:

```
XilInterpolationTable table;
float* data;

data = xil_interpolation_table_get_data(table);
```

SEE ALSO

xil_interpolation_table_create(3), **xil_interpolation_table_destroy**(3), **xil_interpolation_table_get_subsamples**(3), **xil_interpolation_table_get_kernel_size**(3), **xil_state_get_interpolation_tables**(3).

NAME

xil_interpolation_table_get_kernel_size – get the kernel size of the subsample kernels in an interpolation table object

SYNOPSIS

#include <xil/xil.h>

unsigned int xil_interpolation_table_get_kernel_size (XilInterpolationTable *table***);**

DESCRIPTION

xil_interpolation_table_get_kernel_size () gets *kernel size* from the interpolation table object *table.*

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Get kernel size of an interpolation table object:

```
XilInterpolationTable table;
unsigned int kernel_size;

kernel_size = xil_interpolation_table_get_kernel_size(table);
```

SEE ALSO

xil_interpolation_table_create(3), **xil_interpolation_table_destroy**(3), **xil_interpolation_table_get_subsamples**(3), **xil_interpolation_table_get_data**(3), **xil_state_get_interpolation_tables**(3).

NAME

xil_interpolation_table_get_subsamples – get the number of subsamples in an interpolation table object

SYNOPSIS

#include <xil/xil.h>

unsigned int xil_interpolation_table_get_subsamples (XilInterpolationTable *table***);**

DESCRIPTION

xil_interpolation_table_get_subsamples () gets *subsamples* from the interpolation table object *table*. Subsamples refer to the number of divisions between pixels in the source image. Subsampling is used when the reverse mapping from destination pixel to source pixel falls between two source pixels.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Get subsamples of an interpolation table object:

```
XilInterpolationTable table;
unsigned int subsamples;

subsamples = xil_interpolation_table_get_subsamples(table);
```

SEE ALSO

xil_interpolation_table_create(3), **xil_interpolation_table_destroy**(3), **xil_interpolation_table_get_kernel_size**(3), **xil_interpolation_table_get_data**(3), **xil_state_get_interpolation_tables**(3).

NAME

xil_kernel_create, xil_kernel_create_copy, xil_kernel_destroy – create and destroy kernels

SYNOPSIS

```
#include <xil/xil.h>

XilKernel xil_kernel_create (XilSystemState State,
        unsigned int width,
        unsigned int height,
        unsigned int key_x,
        unsigned int key_y,
        float *data);

XilKernel xil_kernel_create_copy (XilKernel kernel);

void xil_kernel_destroy (XilKernel kernel);
```

DESCRIPTION

These routines create and destroy *XilKernel* objects. Kernels are used in image convolution, error diffusion, painting, and band combine operations. The key values specify the key pixel position - a position relative to the upper left corner of the kernel. The key pixel aligns with the output pixel and constrains which input pixels are used to generate the output. Kernel data is single-precision floating point.

xil_kernel_create() creates an *XilKernel* object of the specified size and with the specified data.

xil_kernel_create_copy() creates and returns a copy of the specified kernel. The name of a copy is initially empty (NULL).

xil_kernel_destroy() destroys the specified kernel.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Create a 3x3 kernel for edge-sharpening, with the key value located at the center of the kernel:

```
XilSystemState State;
unsigned int width=3, height=3, key_x=1, key_y=1;
XilKernel kernel;
float data[]={ 0., -1.,  0.,
              -1.,  5., -1.,
               0., -1.,  0. };

kernel = xil_kernel_create (State, width, height, key_x, key_y, data);
```

NOTES

The key pixel must lie within the boundaries of the kernel.

SEE ALSO

xil_convolve(3), **xil_kernel_get_height**(3), **xil_kernel_get_width**(3), **xil_kernel_get_key_x**(3), **xil_kernel_get_key_y**(3), **xil_error_diffusion**(3), **xil_paint**(3), **xil_band_combine**(3).

NAME

xil_kernel_get_by_name, xil_kernel_get_name, xil_kernel_set_name – get and set a kernel object name and get a handle to a kernel by specifying its name

SYNOPSIS

#include <xil/xil.h>

XilKernel xil_kernel_get_by_name (XilSystemState *State***,**
char ∗*name***);**

char∗ **xil_kernel_get_name (XilKernel** *kernel***);**

void xil_kernel_set_name (XilKernel *kernel***,**
char ∗*name***);**

DESCRIPTION

Use these functions to assign names to kernel objects, to read kernel names, and to retrieve kernel objects by name. A predefined kernel is created at the time of an **xil_open**(3) call. This kernel can be retrieved by **xil_kernel_get_by_name**().

xil_kernel_get_by_name() returns the handle to the kernel with the specified name *name.* If such a kernel does not exist, NULL is returned. **xil_kernel_get_by_name**() does not make a copy of the kernel.

xil_kernel_get_name() returns a copy of the specified kernel's name. A call to **free** (3) should be used to free the space allocated by **xil_kernel_get_name**(). If the specified kernel has no name, NULL is returned.

xil_kernel_set_name() sets the name of the specified kernel to the one provided.

Standard Kernel Provided

The XIL library creates a predefined kernel at the time of an **xil_open**(3) call. This kernel, "floyd-steinberg", can be used with error diffusion operations.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Create an edge-sharpening kernel named "sharp1":

```
XilSystemState State;
XilKernel kernel;
float data[] =      { 0.0 -1.0  0.0
                       -1.0  5.0 -1.0
                        0.0 -1.0  0.0 };

kernel = xil_kernel_create(State,3,3,0,0,data);
xil_kernel_set_name(kernel, "sharp1");
```

Use a kernel named "sharp1" to convolve an image:

```
XilSystemState State;
XilImage src, dst;
XilKernel kernel;

kernel = xil_kernel_get_by_name(State,"sharp1");
xil_convolve(src, dst, kernel, XIL_EDGE_ZERO_FILL);
```

NOTES

The set of standard objects is generated for each instantiation of an *XilSystemState*. If these standard objects are deleted, they become unavailable for the duration of the current XIL session.

If you give two kernels the same name, it is not defined which kernel will be retrieved by a call to **xil_kernel_get_by_name**().

SEE ALSO

xil_open(3), **xil_kernel_create**(3).

NAME

xil_kernel_get_height, xil_kernel_get_width, xil_kernel_get_key_x, xil_kernel_get_key_y – read attributes of kernels

SYNOPSIS

#include <xil/xil.h>

unsigned int xil_kernel_get_height (XilKernel *kernel***);**

unsigned int xil_kernel_get_width (XilKernel *kernel***);**

unsigned int xil_kernel_get_key_x (XilKernel *kernel***);**

unsigned int xil_kernel_get_key_y (XilKernel *kernel***);**

DESCRIPTION

These routines read the attributes of *XilKernel* kernel objects. Kernels are used in image convolution, error diffusion, painting, and band combine operations. The key values specify the key pixel position - a position relative to the upper left corner of the kernel. The key pixel aligns with the output pixel and constrains which input pixels are used to generate the output.

xil_kernel_get_height() gets the height of a kernel.

xil_kernel_get_width() gets the width of a kernel.

xil_kernel_get_key_x() gets the *x* coordinate of the key value of the specified kernel.

xil_kernel_get_key_y() gets the *y* coordinate of the key value of the specified kernel.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Get the coordinates of a kernel's key value:

```
XilKernel kernel;
unsigned int key_x, key_y;

key_x = xil_kernel_get_key_x (kernel);
key_y = xil_kernel_get_key_y (kernel);
```

SEE ALSO

xil_convolve(3), **xil_kernel_create**(3), **xil_kernel_create_copy**(3), **xil_kernel_destroy**(3), **xil_error_diffusion**(3), **xil_paint**(3), **xil_band_combine**(3).

NAME

xil_lookup – pass an image through a lookup table.

SYNOPSIS

#include <xil/xil.h>

```
void xil_lookup (XilImage src,
        XilImage dst,
        XilLookup lookup);
```

DESCRIPTION

This routine passes the source image through a lookup table and writes the result into the destination image. The parameters *src* and *dst* are handles to the source and destination images. The source and destination can be different data types. *lookup* is the lookup table.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Pass an image through a lookup table:

```
#define BITSIZE 0x2

XilImage image, retained_image;
XilLookup lookup;
Xil_unsigned8 lookupdata[] = {0, 0, 0, 255, 255, 255};

retained_image = xil_create(state, width, height, 3, XIL_BYTE);
lookup = xil_lookup_create(state, XIL_BIT, XIL_BYTE, 3,
                           BITSIZE, 0, lookupdata);
xil_lookup(image, retained_image, lookup);
```

NOTES

Input and output of the entries in the lookup table must be the same data types as the source and destination images, respectively.

SEE ALSO

xil_lookup_create(3), **xil_lookup_create_combined**(3), **xil_lookup_destroy**(3).

NAME

xil_lookup_convert – calculate a conversion lookup table between a source and destination lookup table

SYNOPSIS

#include <xil/xil.h>

XilLookup xil_lookup_convert (XilLookup *lut1***,**
        XilLookup *lut2***);**

DESCRIPTION

This function calculates a lookup table that converts between the two lookup tables *lut1* and *lut2.* The resulting lookup table's input data type will be the input data type of *lut1,* and its output data type will be the input data type of *lut2.* The lookup table's offset and number of entries are the same as those for *lut1.* Index N of the resulting lookup table contains the index of the nearest color in *lut2* to the color at index N in *lut1.* Nearest color is determined by Euclidean distance. Source and destination lookup tables must have the same input data types, output data types, and number of bands.

This function can be useful when you have an image with a lookup table (and colormap) that contains a relatively small number of values over a wide range. You would first compress the values in the lookup table into a smaller range by using **xil_squeeze_range**(3). Then, to create a colormap that matched your newly compressed lookup table, you would use **xil_lookup_convert**().

RETURN VALUES

NULL	If function fails

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Calculate a lookup table to convert between two lookup tables:

```
XilLookup lut1, lut2, lut3;

lut3 = xil_lookup_convert(lut1, lut2);
```

NOTES

This function cannot be used on combined lookup tables.

SEE ALSO

xil_lookup_create(3), **xil_squeeze_range**(3).

NAME

xil_lookup_create, xil_lookup_create_copy, xil_lookup_destroy – create or destroy lookup tables

SYNOPSIS

#include <xil/xil.h>

XilLookup xil_lookup_create (XilSystemState *State***,**
    XilDataType *input_datatype***,**
    XilDataType *output_datatype***,**
    unsigned int *output_nbands***,**
    unsigned int *num_entries***,**
    short *first_entry_offset***,**
    void **data***);**

XilLookup xil_lookup_create_copy (XilLookup *lookup***);**

void xil_lookup_destroy (XilLookup *lookup***);**

DESCRIPTION

These routines create and destroy lookup tables. Lookup tables are used in transforming data, and specialized lookup tables are used as colormap attributes of images.

xil_lookup_create () creates a lookup table for one band of input data. It can be used to create a single lookup table for converting a single-band input image to a single-band or multiband destination image. Or it can be used to create *n* single lookup tables for a multiband input image with *n* bands; when used for multiband input images, the single lookups created for the input bands must be combined into a *combined* lookup table by calling the **xil_lookup_create_combined**(3) function.

When used to convert a single-band input image to a multiband image, the lookup table must have multiple output data elements per input value; the number of elements must match the number of *output_nbands* specified. When used for converting a single band of input data, the lookup table can have only one output data element per input value, and the destination *output_nbands* must equal 1.

Regardless of whether it is created for single-band or multiband input data, a lookup table allows an offset that describes the input value corresponding to the first table value. Table data can represent any of the allowed image data types, but 1-bit data is stored in an unpacked format as the least significant bit in an 8-bit entry. The tables created for multiband input data can use different offsets, but they must all use the same data types.

The maximum number of entries allowed in the lookup table is determined by the input data type and by the *first_entry_offset*, as specified in the **xil_lookup_create** () call. This ensures that inaccessible lookup table entries are not created. Lookup tables with a *first_entry_offset* of 0 and an input data type of XIL_BYTE may have at most 256 entries. Lookup tables with a *first_entry_offset* of -32768 and an input data type of XIL_SHORT may have at most 65536 entries. Lookup tables with a *first_entry_offset* of 0 and a data type of XIL_SHORT may have at most 32768 entries. This function accepts NULL as a valid value for any of its arguments.

xil_lookup_create_copy () returns a copy of the specified lookup table. Copies of lookup objects have the same *XilVersion* number as the original lookup object. The name of a copy is initially empty (NULL).

xil_lookup_destroy () destroys the specified lookup table. For multiband input data, the tables created for each input band must be destroyed individually; the combined table must also be destroyed.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Create a lookup table for converting an 8-bit pseudocolor image to a 24-bit color image given the colormap components *red, green, blue:*

```
XilSystemState State;
XilLookup lookup_table;
Xil_unsigned8 red[256];          /* red component of colormap */
Xil_unsigned8 green[256];        /* green component of colormap */
Xil_unsigned8 blue[256];         /* blue  component of colormap */
Xil_unsigned8 data[256*3];       /* lookup table data */
int i, j;

for(j=0,i=0; i<256; i++, j+=3) {
   data[j]     = blue[i];
   data[j+1] = green[i];
   data[j+2] = red[i];
}
lookup_table = xil_lookup_create (State, XIL_BYTE, XIL_BYTE, 3, 256, 0, data);
```

SEE ALSO

xil_lookup(3), **xil_lookup_convert**(3), **xil_lookup_create_combined**(3), **xil_lookup_get_input_datatype**(3), **xil_lookup_get_num_entries**(3), **xil_lookup_get_offset**(3), **xil_lookup_get_band_lookup**(3), **xil_lookup_get_output_datatype**(3), **xil_lookup_get_input_nbands**(3), **xil_lookup_get_output_nbands**(3), **xil_lookup_get_colorcube**(3), **xil_lookup_set_offset**(3), **xil_lookup_get_colorcube_info**(3), **xil_lookup_set_values**(3).

NAME

xil_lookup_create_combined – create combined lookup tables

SYNOPSIS

#include <xil/xil.h>

XilLookup xil_lookup_create_combined (XilSystemState *State***,**
XilLookup *lookup_list[]***,**
unsigned int *num_lookups***)**

DESCRIPTION

xil_lookup_create_combined () creates a combined lookup table. A combined lookup table is used for transforming multiband data to multiband data. Compare this function with **xil_lookup_create**(3), which converts single-band data to single-band or multiband data.

Combined lookups are a combination of *n* single lookup tables, where *n* is the number of bands in the input image you want to convert. Each single lookup must be a 1-band to 1-band lookup table; the tables must all have the same data type, but each can use a different offset.

To create a lookup table for a multiband input image, you call **xil_lookup_create** (3) once for each band in the input image, then combine the single lookup tables into a *combined* lookup table by calling **xil_lookup_create_combined** ().

xil_lookup_create_combined () returns a handle to a data structure of type *XilLookup,* which is the combined lookup. The parameter *State* is a handle to the system-state data structure created when you initialize the XIL library, *lookup_list[]* is an array of type *XilLookup* that stores the single lookup tables created for each of the input image's bands, and *num_lookups* indicates how many lookup tables are stored in the *lookup_list[]* array.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Create a combined lookup table for converting a 24-bit color image to another 24-bit color image whose green band is accented but whose red and blue bands are subdued:

```
XilSystemState State;
XilLookup lookup_tables[3];
XilLookup combined_lookup_table;
Xil_unsigned8 red[256];          /* red component of lookup */
Xil_unsigned8 green[256];        /* green component of lookup */
Xil_unsigned8 blue[256];         /* blue  component of lookup */
int i;

for(i=0; i<256; i++) {
    green[i] = (i + 20) < 255 ? i + 20 : 255;
     blue[i] = red[i] = (i - 10) < 0 ? 0 : i - 10;
}
lookup_tables[0] = xil_lookup_create(State, XIL_BYTE, XIL_BYTE,
                        1, 256, 0, red);
lookup_tables[1] = xil_lookup_create(State, XIL_BYTE, XIL_BYTE,
                        1, 256, 0, green);
lookup_tables[2] = xil_lookup_create(State, XIL_BYTE, XIL_BYTE,
                        1, 256, 0, blue);
combined_lookup_table = xil_lookup_create_combined(State,
                                    lookup_tables, 3);
```

SEE ALSO

xil_lookup(3), **xil_lookup_create**(3), **xil_lookup_convert**(3), **xil_lookup_get_band_lookup**(3), **xil_lookup_get_input_nbands**(3), **xil_lookup_get_input_datatype**(3), **xil_lookup_get_num_entries**(3), **xil_lookup_get_offset**(3), **xil_lookup_get_output_datatype**(3), **xil_lookup_get_output_nbands**(3), **xil_lookup_get_colorcube**(3), **xil_lookup_set_offset**(3), **xil_lookup_get_colorcube_info**(3), **xil_lookup_set_values**(3).

NAME

xil_lookup_get_band_lookup – get a single lookup table out of a combined lookup

SYNOPSIS

#include <xil/xil.h>

XilLookup xil_lookup_get_band_lookup (XilLookup *lookup***,**
    unsigned int *band_num***)**

DESCRIPTION

This function creates a copy of the lookup for the specified band in a combined lookup table. *lookup* is the handle to the combined lookup table, and *band_num* is the band number to be copied.

The lookup table that is returned is a single lookup table with one output element per input value. It can be used to convert a single-band input image to another single-band input image, or it can be used as the lookup table for one band of a multiband input image. It cannot be used to convert single-band data to multiband data.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Get a copy of the lookup table in the first band of a combined lookup table built for converting a 24-bit color image to another 24-bit color image:

```
XilLookup band1_lookup;
XilLookup combined_lookup_table;

band1_lookup = xil_lookup_get_band_lookup(combined_lookup_table, 0);
```

SEE ALSO

xil_lookup(3), **xil_lookup_create**(3), **xil_lookup_create_combined**(3), **xil_lookup_convert**(3), **xil_lookup_get_input_nbands**(3), **xil_lookup_get_input_datatype**(3), **xil_lookup_get_num_entries**(3), **xil_lookup_get_offset**(3), **xil_lookup_get_output_datatype**(3), **xil_lookup_get_output_nbands**(3), **xil_lookup_get_colorcube**(3), **xil_lookup_set_offset**(3), **xil_lookup_get_colorcube_info**(3), **xil_lookup_set_values**(3).

NAME

xil_lookup_get_by_name, xil_lookup_get_name, xil_lookup_set_name – get and set a lookup table name and get a handle to a lookup table by specifying its name

SYNOPSIS

#include <xil/xil.h>

XilLookup xil_lookup_get_by_name (XilSystemState *State***,**
    char ∗*name***);**

char∗ **xil_lookup_get_name (XilLookup** *lookup***);**

void xil_lookup_set_name (XilLookup *lookup***,**
    char ∗*name***);**

DESCRIPTION

Use these functions to assign names to lookup tables, retrieve lookup tables by name, and get the handle of a lookup table by specifying its name. Some predefined lookup tables are created at the time of an **xil_open**(3) call. These lookup tables can be retrieved by **xil_lookup_get_by_name** ().

xil_lookup_get_by_name () returns the handle to the lookup table with the specified name *name.* If such a lookup table does not exist, NULL is returned. **xil_lookup_get_by_name** () does not make a copy of the lookup table.

xil_lookup_get_name () returns a copy of the specified lookup table's name. A call to **free** (3) should be used to free the space allocated by **xil_lookup_get_name** (). If the specified lookup table has no name, NULL is returned.

xil_lookup_set_name () sets the name of the specified lookup table *name.*

Standard Lookup Tables Provided

The XIL library creates several predefined lookup tables at the time of an **xil_open**(3) call. The names of these lookup tables and their suggested uses follow.

Lookup Table Name	*Suggested Use*
"yuv_to_rgb"	RGB lookup table for displaying 8:5:5 dithered YCC data
"cc855"	A good colorcube for dithering YCC data into 200 colors. This lookup table is created with an offset of 54.
"cc496"	A good colorcube for dithering RGB data into 216 colors. This lookup table is created with an offset of 38.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Create an inverse 8-bit lookup table named "invert":

```
XilSystemState State;
XilLookup lookup;
int i;
unsigned char data[256];

for (i=0; i<256; i++) data[i] = 255 - i;
lookup = xil_lookup_create(State,XIL_BYTE,XIL_BYTE,1,256,0,data);
xil_lookup_set_name(lookup,"invert");
```

Use a lookup table named "invert" to remap an image:

```
XilSystemState State;
XilImage src, dst;
XilLookup lookup;

lookup = xil_lookup_get_by_name(State,"invert");
xil_lookup(src, dst, lookup);
```

NOTES

The set of standard objects is generated for each instantiation of an *XilSystemState*. If these standard objects are deleted, they become unavailable for the duration of the current XIL session.

If you give two lookup tables the same name, it is not defined which lookup table will be retrieved by a call to **xil_lookup_get_by_name** ().

SEE ALSO

xil_open(3), **xil_lookup_create**(3), **xil_lookup_create_combined**(3).

NAME

xil_lookup_get_input_datatype, xil_lookup_get_num_entries, xil_lookup_get_offset, xil_lookup_get_output_datatype, xil_lookup_get_input_nbands, xil_lookup_get_output_nbands, xil_lookup_set_offset – operations on lookup tables

SYNOPSIS

#include <xil/xil.h>

XilDataType xil_lookup_get_input_datatype (XilLookup *lookup***);**

unsigned int xil_lookup_get_num_entries (XilLookup *lookup***);**

short xil_lookup_get_offset (XilLookup *lookup***);**

XilDataType xil_lookup_get_output_datatype (XilLookup *lookup***);**

unsigned int xil_lookup_get_input_nbands (XilLookup *lookup***);**

unsigned int xil_lookup_get_output_nbands (XilLookup *lookup***);**

void xil_lookup_set_offset (XilLookup *lookup***,**
short *offset***);**

DESCRIPTION

These routines read and set the values of lookup table attributes. Lookup tables are used in transforming data. Lookup tables used for single-band input images can have multiple output data per input value. Lookup tables allow an offset that describes the input value corresponding to the first table value.

Table data can represent any of the allowed image data types, but 1-bit data is stored in an unpacked format as the least significant bit in an 8-bit entry.

xil_lookup_get_input_datatype () gets the data type of the expected input to the lookup table.

xil_lookup_get_num_entries () gets the number of entries in the lookup table. This function cannot be used on combined lookup tables.

xil_lookup_get_offset () returns the offset value used to map the lookup table index to a pixel value of a particular data type. The offset value is added to a lookup table index to return a pixel value, and subtracted from a pixel value to return an index into the lookup table. This function cannot be used on combined lookup tables.

For example, if a lookup table has an offset of 16, then entry 0 in the lookup table maps to an actual value of 16, entry 1 maps to 17, and so on. Therefore, if you wanted to find the RGB value for pixel 36, you would take lookup table entry 20 (pixel value 36 minus offset value 16).

xil_lookup_get_output_datatype () gets the data type of the expected output from the lookup table.

xil_lookup_get_input_nbands () gets the number of bands expected in the input.

xil_lookup_get_output_nbands () gets the number of bands expected in the output.

xil_lookup_set_offset () sets the offset value to the one specified. This function cannot be used on combined lookup tables.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Calculate the buffer size (in bytes) necessary to hold all the values in a lookup table:

```
XilLookup lookup_table;
unsigned int nbands;
XilDataType datatype;
unsigned int num_entries;
long buffer_size;

nbands = xil_lookup_get_output_nbands (lookup_table);
datatype = xil_lookup_get_output_datatype (lookup_table);
num_entries = xil_lookup_get_num_entries (lookup_table);

switch (datatype) {
  case XIL_BIT:
  case XIL_BYTE:
    buffer_size = nbands * num_entries * sizeof(Xil_unsigned8);
    break;
  case XIL_SHORT:
    buffer_size = nbands * num_entries * sizeof(Xil_signed16);
    break;
}
```

SEE ALSO

xil_lookup_create(3), **xil_lookup_create_combined**(3), **xil_lookup_create_copy**(3), **xil_lookup_destroy**(3), **xil_lookup_convert**(3), **xil_lookup_get_band_lookup**(3), **xil_lookup_set_values**(3).

NAME

xil_lookup_get_version – get a unique version number for a lookup table

SYNOPSIS

#include <xil/xil.h>

XilVersionNumber xil_lookup_get_version (XilLookup *lookup***);**

DESCRIPTION

This function gets a unique identifier associated with a lookup table. This identifier changes whenever the lookup table values change. Unchanged copies created with **xil_lookup_create_copy**(3) will have the same version number as the original.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Get a version number for a lookup table:

```
XilVersionNumber version1, version2;
XilLookup table1, table2;

version1=xil_lookup_get_version(table1);
table2=xil_lookup_create_copy(table1);
version2=xil_lookup_get_version(table2);
if (version1!=version2)
        printf("Error in lookup copy, different verson numbers.\n");
```

SEE ALSO

xil_lookup_create(3), **xil_lookup_create_combined**(3), **xil_lookup_create_copy**(3), **xil_lookup_destroy**(3), **xil_lookup_get_input_datatype**(3), **xil_lookup_get_num_entries**(3), **xil_lookup_get_offset**(3), **xil_lookup_get_output_datatype**(3), **xil_lookup_get_input_nbands**(3), **xil_lookup_get_output_nbands**(3), **xil_lookup_get_band_lookup**(3), **xil_lookup_set_offset**(3), **xil_lookup_convert**(3), **xil_lookup_get_values**(3), **xil_lookup_set_values**(3).

NAME

xil_lookup_set_values, xil_lookup_get_values – set and get values in a lookup table

SYNOPSIS

```
#include <xil/xil.h>

void xil_lookup_set_values ( XilLookup lookup,
        short start,
        unsigned int num_values,
        void *data);

void xil_lookup_get_values ( XilLookup lookup,
        short start,
        unsigned int num_values,
        void *data);
```

DESCRIPTION

xil_lookup_set_values () sets the specified values in the lookup table to those in *data.* The version number of the lookup table is updated whenever this is done.

xil_lookup_get_values () copies *num_values* lookup table values into the user-supplied buffer *data. start* is the table entry position at which to begin reading values. The user is responsible for allocating and freeing the buffer. The example below shows how big to make the buffer.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Get a sequence of data values out of a lookup table containing XIL_SHORT data values, and add 1 to each entry:

```
XilLookup table;
unsigned int count, buf_size, i, j, output_nbands;
short start;
void* buffer;

/* extract 100 entries starting at the 42nd value in the table /*
        /* (assumes a table offset of 0 )*/
count = 100; start = 42;

/* determine how big to make the values buffer (assume XIL_SHORT datatype) */
output_nbands = xil_lookup_get_output_nbands (table);
buf_size = output_nbands * count * sizeof(Xil_signed16);

/* allocate the values buffer */
buffer = (void *) malloc (buf_size);

/* get the current values in the lookup table */
xil_lookup_get_values (table, start, count, buffer);
```

```
/* increment all the extracted values by 1 */
pixel_ptr = (Xil_signed16 *) buffer;
for (i = 0; i < count; i++)
   for (j = 0; j < output_nbands; j++) {
     *pixel_ptr += 1;
      pixel_ptr++;
   }

/* replace the values in the lookup table */
xil_lookup_set_values (table, start, count, buffer);
```

NOTES

These functions cannot be used on combined lookup tables.

SEE ALSO

xil_lookup_create(3), **xil_lookup_create_copy**(3), **xil_lookup_destroy**(3), **xil_lookup_get_input_datatype**(3), **xil_lookup_get_num_entries**(3), **xil_lookup_get_offset**(3), **xil_lookup_get_output_datatype**(3), **xil_lookup_get_output_nbands**(3), **xil_lookup_set_offset**(3), **xil_lookup_convert**(3), **xil_lookup_get_version**(3).

NAME

xil_max – find the larger of pixels in two images

SYNOPSIS

#include <xil/xil.h>

void xil_max (XilImage *src1***,**
        XilImage *src2***,**
        XilImage *dst***);**

DESCRIPTION

xil_max () performs a pixel-by-pixel max() operation of the *src1* and *src2* images and stores the result in the *dst* (destination) image.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Find the larger of *image1* and *image2* and store the result in *dst* :

```
XilImage image1, image2, dst;

xil_max(image1, image2, dst);
```

NOTES

Source and destination images must be of the same data type and have the same number of bands. In-place operations are supported.

SEE ALSO

xil_extrema(3)

NAME

xil_min – find the lesser of pixels in two images

SYNOPSIS

#include <xil/xil.h>

void xil_min (XilImage *src1***,**
XilImage *src2***,**
XilImage *dst***);**

DESCRIPTION

xil_min () performs a pixel-by-pixel min() operation of the *src1* and *src2* images and stores the result in the *dst* (destination) image.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Find the lesser of *image1* and *image2* and store the result in *dst* :

```
XilImage image1, image2, dst;

xil_min(image1, image2, dst);
```

NOTES

Source and destination images must be of the same data type and have the same number of bands. In-place operations are supported.

SEE ALSO

xil_extrema(3)

NAME

xil_multiply, xil_multiply_const – image multiplication operations.

SYNOPSIS

#include <xil/xil.h>

void xil_multiply (XilImage *src1***,**
    XilImage *src2***,**
    XilImage *dst***);**

void xil_multiply_const (XilImage *src1***,**
    float **constants***,**
    XilImage *dst***);**

DESCRIPTION

xil_multiply () performs a pixel-by-pixel multiplication of the *src1* image by the *src2* image and stores the result in the *dst* (destination) image. If the result of the operation is out of range for a particular data type, the result is clamped to the minimum or maximum value for the data type. Results for XIL_BYTE operations, for example, are clamped to 0 if they are less than 0 and 255 if they are greater than 255.

xil_multiply_const () performs a pixel-by-pixel multiplication of the *src1* image by the *constants* values and stores the result in the *dst* (destination) image. For an n-band image, n float values must be provided, one per band. The values in band 0 are multiplied by the value the first element of the *constants* array, and so on. Pixel values are rounded and clipped according to image data type.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Multiply *image2* by *image1* and store the result in *dst* :

```
XilImage image1, image2, dst;

xil_multiply(image1, image2, dst);
```

Multiply 4-band *image1* by *constants* and store the result in *dst* :

```
XilImage image1, dst;
float constants[4];

constants[0] = 1.0;
constants[1] = 2.0;
constants[2] = 0.5;
constants[3] = 2.0;
xil_multiply_const(image1, constants, dst);
```

NOTES

Source and destination images must be the same data type and have the same number of bands. In-place operations are supported.

NAME

xil_nearest_color – find nearest match of pixel values to entries in colormap

SYNOPSIS

```
#include <xil/xil.h>

void xil_nearest_color (XilImage src,
        XilImage dst,
        XilLookup cmap);
```

DESCRIPTION

This routine performs a pixel-by-pixel search for the nearest matching color in the supplied lookup table and sets the destination image pixel value to the appropriate colormap index. Nearest color is determined by calculating Euclidean distance for n-bands. *src* is the source image. *dst* is the destination image. *cmap* is a lookup table with the number of output bands equal to the number of bands in the source image.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Match nearest color for a 3-band image:

```
XilImage src;                   /* 3-band source image */
XilImage dst;                   /* 1-band destination image */
XilLookup colormap;             /* colormap */

xil_nearest_color(src, dst, colormap);
```

NOTES

The source image must have the same data type and the same number of bands as the lookup table. The destination image must have the same data type as the lookup table's input data type.

SEE ALSO

xil_colorcube_create(3), **xil_lookup_create**(3), **xil_lookup_get_by_name**(3).

NAME

xil_not – bitwise logical NOT operation.

SYNOPSIS

#include <xil/xil.h>

void xil_not (XilImage *src***,**
XilImage *dst***);**

DESCRIPTION

This function performs a bitwise logical NOT operation on each pixel of the *src* (source) image and stores the results in the *dst* (destination) image.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Bitwise logical NOT *image1* and store the result in *dst* :

```
XilImage image1, dst;

xil_not(image1 dst);
```

NOTES

Source and destination images must be the same data type and have the same number of bands. In-place operations are supported.

NAME

xil_open, xil_close – open and close an XIL session

SYNOPSIS

#include <xil/xil.h>

XilSystemState xil_open ();

void xil_close (XilSystemState *State***);**

DESCRIPTION

xil_open () is used to begin an XIL session. It must be called before any other XIL routine. A single *XilSystemState* object is created and returned when **xil_open** () is invoked. If the function is successful, a handle to the *XilSystemState* object is returned. This object can only be destroyed by a subsequent call to **xil_close** () using the specified handle.

When **xil_open** () is called, the XIL library creates a predefined kernel object, four predefined dither mask objects, three predefined look-up tables, and ten predefined color space objects. These objects can be retrieved by using the relevant XIL "get by name" function. For example, you would use **xil_kernel_get_by_name**(3) to retrieve the predefined kernel. Consult the relevant manual pages for more details.

xil_close () is used to end an XIL session. A handle to the *XilSystemState* object describing the session to be terminated is passed to the function. The *XilSystemState* object and all resources associated with that XIL session are destroyed, and the XIL session is terminated.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Open and close the XIL Library:

```
XilSystemState State;
State = xil_open();
xil_close(State);
```

NOTES

Multiple calls to **xil_open**() produce completely separate system states that provide completely separate XIL environments. Objects created in one environment cannot be used in another environment. This feature is intended to allow layered software that uses the XIL library to be completely independent from all other layered software that uses the XIL library.

If your program creates a display image and you do not destroy the image with **xil_destroy**(), you must close the XIL library (with **xil_close**()) before you disconnect your program from the X server and display.

SEE ALSO

xil_create(3), **xil_cis_create**(3), **xil_kernel_create**(3), **xil_lookup_create**(3), **xil_roi_create**(3), **xil_sel_create**(3), **xil_kernel_get_by_name**(3), **xil_lookup_get_by_name**(3), **xil_dithermask_get_by_name**(3), **xil_colorspace_get_by_name**(3).

NAME

xil_or, xil_or_const – bitwise logical OR operations

SYNOPSIS

```
#include <xil/xil.h>

void xil_or (XilImage src1,
        XilImage src2,
        XilImage dst);

void xil_or_const (XilImage src1,
        unsigned int *constants,
        XilImage dst);
```

DESCRIPTION

xil_or () performs a bitwise logical OR operation on each pixel of the *src2* (source) image with the corresponding pixel in the *src1* image and stores the result in the *dst* (destination) image.

xil_or_const () performs a bitwise logical OR operation on each pixel of the *src1* (source) image with the *constants* values and stores the results in the *dst* (destination) image. For a n-band image, n unsigned integers must be provided, one per band. The values in band 0 ar ORed with the value in *constants[0],* and so on.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Bitwise logical OR *image2* with *image1* and store the result in *dst:*

```
XilImage image1, image2, dst;

xil_or(image1, image2, dst);
```

Bitwise logical OR a 4-band *image1* with 4 different constants and store the result in *dst:*

```
XilImage image, dst;
unsigned int constants[4];

constants[0] = 1;
constants[1] = 1;
constants[2] = 1;
constants[3] = 0;
xil_or_const(image, constants, dst);
```

NOTES

Source and destination images must be the same data type and have the same number of bands. In-place operations are supported.

NAME

xil_ordered_dither – use ordered dithering to convert a multiband or single-band image into a single-band image with a colormap

SYNOPSIS

#include <xil/xil.h>

void xil_ordered_dither(XilImage *src***,**
    XilImage *dst,*
    XilLookup *cmap,*
    XilDitherMask *mask)*

DESCRIPTION

This routine performs an ordered dithering of a *src* (source) image with dither matrices and produces a single-band *dst* (destination) image. *cmap* is a lookup table and must be a colorcube. *mask* is a dither mask and must contain n matrices for an n-band source image. These matrices must have the same dimensions and contain floating point values between 0.0 and 1.0.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Ordered-dither a 3-band image into a single-band image using a 4x4 dither mask:

```
XilImage src;                    /* 3-band source image */
XilImage dst;                    /* 1-band destination image */
XilLookup colormap;              /* colorcube */
XilDitherMask dithermask;        /* 3 dither matrices */
float data[]= {  0.0,    0.5,    0.125,  0.625,
                 0.75,   0.25,   0.875,  0.375,
                 0.1875, 0.6875, 0.0625, 0.5625,
                 0.9375, 0.4375, 0.8125, 0.3125,
                 0.0,    0.5,    0.125,  0.625,
                 0.75,   0.25,   0.875,  0.375,
                 0.1875, 0.6875, 0.0625, 0.5625,
                 0.9375, 0.4375, 0.8125, 0.3125,
                 0.0,    0.5,    0.125,  0.625,
                 0.75,   0.25,   0.875,  0.375,
                 0.1875, 0.6875, 0.0625, 0.5625,
                 0.9375, 0.4375, 0.8125, 0.3125};

dithermask = xil_dithermask_create(State, 4, 4, 3, data);

xil_ordered_dither(src, dst, colormap, dithermask);
```

NOTES

In-place operations can occur when converting a single-band image into a single-band image of the same data type with a colormap.

SEE ALSO

xil_dithermask_create(3), **xil_lookup_create_copy**(3), **xil_lookup_destroy**(3), **xil_lookup_get_input_datatype**(3), **xil_lookup_get_num_entries**(3), **xil_lookup_get_offset**(3), **xil_lookup_get_output_datatype**(3), **xil_lookup_get_output_nbands**(3), **xil_lookup_set_offset**(3), **xil_lookup_convert**(3), **xil_colorcube_create**(3), **xil_lookup_get_colorcube**(3), **xil_lookup_get_colorcube_info**(3).

NAME

xil_paint – perform paint on specified point list

SYNOPSIS

#include <xil/xil.h>

void xil_paint (XilImage *src***,**
    XilImage *dst***,**
    float **color***,**
    XilKernel *brush***,**
    unsigned int *count***,**
    float **coord_list***);**

DESCRIPTION

This function blends portions of an image with a single color using a 2-D brush. The brush is applied for each point in a list of coordinates. For each entry in the brush, the associated pixel in the image is colored. *src* is the source image handle. *dst* is the destination image handle. *color* is a pointer to the floating-point array that specifies the brush color [0...(nbands-1)] for each pixel. *brush* is a kernel with values between 0.0 and 1.0.

The destination value is determined by this equation:

dst_pixel = (brush_value ∗ color) + ((1.0 - brush_value) ∗ src_pixel)

Where the brush value is 0.0, the destination value is the source value. Where the brush value is 1.0, the destination value is the paint color.

count is the count of *x,y* coordinate pairs. *coord_list* is a pointer to the floating-point array that specifies the *x,y* coordinate pairs.

ROI Behavior

This function performs the paint operation in the source image on each point in the coordinate list. The painted pixels within the ROI (region of interest) are output to the destination image.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

For this example, the source and destination images contain 2 bands. Create a 2 x 2 brush with the key pixel at the upper left corner of the kernel. Perform paint at pixel (x,y) = (100,75).

```
XilImage src;
XilImage dst;
float paint_color[2] = {127.0, 255.0};
XilKernel brush;
float brush_data[4] = {1.0, 0.5, 0.5, 0.0};
unsigned int count = 1;
float coord_list[2] =  {100.0, 75.0};

brush = xil_kernel_create(system_state,2,2,0,0,brush_data);

xil_paint(src, dst, paint_color, brush, count, coord_list);
```

NOTES

Source and destination images must be the same data type and have the same number of bands. For an n-band image, the array of floating point numbers for *color* must be of size n. Only pixels that are blended with the paint color are output to the destination image. In-place operations are supported.

SEE ALSO

xil_kernel_create(3), **xil_kernel_destroy**(3), **xil_blend**(3).

NAME

xil_rescale – rescale the data in an image

SYNOPSIS

#include <xil/xil.h>

void xil_rescale (XilImage *src***,**
        XilImage *dst***,**
        float **scale***,**
        float **offset***);**

DESCRIPTION

This routine performs a pixel-by-pixel rescaling of the data in a *src* (source) image by first multiplying each pixel value by a scale factor and then adding an offset. The result is stored in the *dst* (destination) image. For an n-band image, each array of constants must contain n floats The values in band 0 are scaled by *scale[0]* and added to *offset[0]*, and so on.

Pixel values are clipped according to image data type. In this function, a floating point intermediate value is calculated, so clipping / rounding is done after both the multiplication and the addition have occurred.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Rescale a 4-band short *src* image into the range of XIL_BYTE, and store the result in *dst:*

```
XilImage src, dst;
float scale_values[4], offset_values[4];

src=xil_create(State, 512, 512, 3, XIL_SHORT);
dst=xil_create(State, 512, 512, 3, XIL_SHORT);

/* scale factors for each band */
scale_values[0] = 127.5/32767.0;
scale_values[1] = 127.5/32767.0;
scale_values[2] = 127.5/32767.0;
scale_values[3] = 127.5/32767.0;

/* offset factors for each band */
offset_values[0] = 127.5;
offset_values[1] = 127.5;
offset_values[2] = 127.5;
offset_values[3] = 127.5;

xil_rescale(src, dst, scale_values, offset_values);
```

NOTES

Source and destination images must be the same data type and have the same number of bands. In-place operations are supported.

NAME

xil_roi_add_image – add a binary image to an ROI

SYNOPSIS

#include <xil/xil.h>

void xil_roi_add_image (XilRoi *roi***,**
XilImage *image***);**

DESCRIPTION

This function adds the specified XIL_BIT image to the specified region of interest (ROI). Bits that are set in the image are added to the region of interest. The image's origin is used to position the image pixels with respect to the ROI. The origin of the ROI is always (0.0, 0.0), corresponding to the upper left corner. The image must be of type XIL_BIT and consist of only one band.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Do a logical AND of two binary images, and add the result to ae region of interest:

```
XilRoi roi;
XilImage  image1, image2, image3;

xil_and (image1, image2, image3);
xil_roi_add_image (roi, image3);
```

SEE ALSO

xil_get_roi(3), **xil_set_roi**(3), **xil_roi_add_rect**(3), **xil_roi_add_region**(3), **xil_roi_create**(3), **xil_roi_create_copy**(3), **xil_roi_destroy**(3), **xil_roi_get_as_image**(3), **xil_roi_get_as_region**(3), **xil_roi_intersect**(3), **xil_roi_subtract_rect**(3), **xil_roi_translate**(3), **xil_roi_unite**(3).

NAME

xil_roi_add_rect – add a rectangle to an ROI

SYNOPSIS

#include <xil/xil.h>

```
void xil_roi_add_rect ( XilRoi roi,
        long x,
        long y,
        long width,
        long height);
```

DESCRIPTION

This function adds the specified rectangle to the specified region of interest (ROI).

The coordinates of the rectangle are with respect to the storage of the image. That is, an ROI coordinate of (0.0, 0.0) always refers to the upper left pixel in an image, regardless of the image's origin.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Add two rectangles to an ROI object, one beginning at (0,0) and ending at (34,94), the other beginning at (10,20) and ending at (109,69):

```
XilRoi roi = xil_roi_create(state);
long  xstart, ystart, width, height;

xstart=0; ystart=0; width=36; height=96;
xil_roi_add_rect (roi, xstart, ystart, width, height);
xstart=10; ystart=20; width=100; height=50;
xil_roi_add_rect (roi, xstart, ystart, width, height);
```

SEE ALSO

xil_get_roi(3), **xil_set_roi**(3), **xil_roi_add_image**(3), **xil_roi_add_region**(3), **xil_roi_create**(3), **xil_roi_create_copy**(3), **xil_roi_destroy**(3), **xil_roi_get_as_image**(3), **xil_roi_get_as_region**(3), **xil_roi_intersect**(3), **xil_roi_subtract_rect**(3), **xil_roi_translate**(3), **xil_roi_unite**(3).

NAME

xil_roi_add_region – add an X region to an ROI

SYNOPSIS

#include <xil/xil.h>

void xil_roi_add_region (XilRoi *roi***,**
    Region *region***);**

DESCRIPTION

This function adds a specified X region to a specified region of interest (ROI).

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Add an X region to an ROI:

```
XilRoi roi;
Region region;

xil_roi_add_region (roi, region);
```

SEE ALSO

xil_get_roi(3), **xil_set_roi**(3), **xil_roi_add_image**(3), **xil_roi_add_rect**(3), **xil_roi_create**(3), **xil_roi_create_copy**(3), **xil_roi_destroy**(3), **xil_roi_get_as_image**(3), **xil_roi_get_as_region**(3), **xil_roi_intersect**(3), **xil_roi_subtract_rect**(3), **xil_roi_translate**(3), **xil_roi_unite**(3).

NAME

xil_roi_create, xil_roi_create_copy, xil_roi_destroy – create or destroy ROIs

SYNOPSIS

#include <xil/xil.h>

XilRoi xil_roi_create (XilSystemState *State***);**

XilRoi xil_roi_create_copy (XilRoi *roi***);**

void xil_roi_destroy (XilRoi *roi***);**

DESCRIPTION

These routines create and destroy region of interest (ROI) objects.

xil_roi_create () creates an *XilRoi* object. It is initially empty. You can use **xil_roi_add_rect**(3), **xil_roi_add_region**(3), or **xil_roi_add_image**(3) to add rectangles to an ROI. ROIs exist in the coordinate system of the image storage. That is, an ROI coordinate of (0.0, 0.0) always refers to the upper left pixel in an image, regardless of the image's origin.

xil_roi_create_copy () returns a copy of the specified ROI. The name of a copy is initially empty (NULL).

xil_roi_destroy () destroys the specified ROI.

XIL and Regions of Interest

ROIs provide a way to limit operations to a specific part of image data. ROIs are attributes of images; they specify what part of an image may be used.

As a destination attribute, the ROI functions as a "write mask" for the destination image. If the ROI is valid (non-zero) for a particular pixel, that pixel may be modified by the operation; otherwise, the pixel is not written.

For a source image, the ROI defines what part of the source image may go toward modifying the destination. In the case of some of the geometric operators, this means a rectangular ROI in the source maps to a nonrectangular area of modification in the destination.

Area operations, such as interpolated geometric zooms, convolution, and erosion, may use data outside the source ROI in creating their output pixel. In the case of geometric operators, the source pixel is generated if the backward-mapped subpixel position lies in the ROI; pixels used in the interpolation may be outside the ROI. For convolution and erosion / dilation, the source pixel is used if it is inside the source ROI; the surrounding pixels used in generating the convolution may be outside the ROI. In the destination, only the output pixel is tested against the destination ROI for writability.

If more than one image in an operation has an ROI attribute, the intersection of all the ROIs (with source ROIs transformed into the destination space) is used to mask the destination.

Although they are image attributes, ROIs attached to an image are not modified along with the image. Destination images retain their own ROIs and do not adopt the ROI of the source image. Copying an image does not copy the ROI attribute; it must be copied explicitly. In addition, creating a child image from an image with an ROI does not cause the child to inherit the portion of the parent's ROI covering it. Installation of an ROI on a child image must be performed explicitly.

The coordinate space of the ROI is conceptually tied to the image storage. That is, the location of the ROI with respect to image data is not changed by changing the image origin.

Operations on ROIs may be performed by retrieving the ROI as a 1-bit image, passing the image to the appropriate XIL operator, then reinstalling the image as an ROI. Several functions exist to operate directly on ROIs without having to first convert them into an external format. This probably provides better performance for these supported operators. A list of ROI operations and their corresponding man pages is given below.

Get an ROI	**xil_get_roi**(3)
Set an ROI	**xil_set_roi**(3)
Add a binary image to an ROI	**xil_roi_add_image**(3)
Add a rectangle to an ROI	**xil_roi_add_rect**(3)
Add an X region to an ROI	**xil_roi_add_region**(3)
Create and return a copy of an ROI	**xil_roi_create_copy**(3)
Destroy an ROI	**xil_roi_destroy**(3)
Get an image version of an ROI	**xil_roi_get_as_image**(3)
Get an X region version of an ROI	**xil_roi_get_as_region**(3)
Find the intersection of two ROIs	**xil_roi_intersect**(3)
Subtract a rectangle from an ROI	**xil_roi_subtract_rect**(3)
Translate an ROI	**xil_roi_translate**(3)
Find the union of two ROIs	**xil_roi_unite**(3)

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Create an ROI, add a rectangle to it (beginning at (10,20) and ending at (109,69), associate it with an image, and then destroy it:

```
XilSystemState State;
XilImage image;
XilRoi roi;
long  xstart=10, ystart=20, width=100, height=50;

roi = xil_roi_create (State);
xil_roi_add_rect (roi, xstart, ystart, width, height);
xil_set_roi (image, roi);
xil_roi_destroy (roi);
```

SEE ALSO

xil_roi_add_image(3), **xil_roi_add_rect**(3), **xil_roi_add_region**(3), **xil_roi_get_as_image**(3), **xil_roi_get_as_region**(3), **xil_roi_intersect**(3), **xil_roi_subtract_rect**(3), **xil_roi_translate**(3), **xil_roi_unite**(3), **xil_get_roi**(3), **xil_set_roi**(3).

NAME

xil_roi_get_as_image – get an image version of an ROI

SYNOPSIS

#include <xil/xil.h>

XilImage xil_roi_get_as_image (XilRoi *roi***);**

DESCRIPTION

This function returns a handle to a new binary (XIL_BIT) image that is an image representation of the supplied ROI. The image returned will be just large enough to contain all of the regions of interest; in other words, a bounding box image is generated. The beginning x and y values for the upper-leftmost ROI are encoded as -(x) and -(y) in the returned image's origin. For example, if the upper-left ROI pixel in the source image is at location (50,50), it is encoded to (-50, -50) in the returned image's origin. If a pixel in the image is contained within the ROI, it is set to 1; otherwise, it is set to 0.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Get the ROI associated with an image. Then create an image mask that corresponds to the ROI returned:

```
XilSystemState State;
XilImage image, image_mask;
XilRoi roi;

roi = xil_get_roi (image);
if (roi == NULL) {
   /* The image had no ROI associated with it,
     create one that encompasses the whole image */
   roi = xil_roi_create (State);
   xil_roi_add_rect (roi, 0, 0, xil_get_width(image), xil_get_height(image));
}
image_mask = xil_roi_get_as_image (roi);
```

SEE ALSO

xil_get_roi(3), **xil_set_roi**(3), **xil_roi_add_image**(3), **xil_roi_add_rect**(3), **xil_roi_add_region**(3), **xil_roi_create**(3), **xil_roi_create_copy**(3), **xil_roi_destroy**(3), **xil_roi_get_as_region**(3), **xil_roi_intersect**(3), **xil_roi_subtract_rect**(3), **xil_roi_translate**(3), **xil_roi_unite**(3).

NAME

xil_roi_get_as_region – get an X region version of an ROI

SYNOPSIS

#include <xil/xil.h>

Region xil_roi_get_as_region (XilRoi *roi***);**

DESCRIPTION

This function returns a handle to an X region that corresponds to the supplied ROI.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Get the ROI associated with an image. Then create an X region that corresponds to the ROI returned:

```
XilSystemState State;
XilImage image;
XilRoi roi;
Region region;

roi = xil_get_roi (image);
if (roi == NULL) {
   /* The image had no ROI associated with it,
      create one that encompasses the whole image */
   roi = xil_roi_create (State);
   xil_roi_add_rect (roi, 0, 0, xil_get_width(image), xil_get_height(image));
}
region = xil_roi_get_as_region (roi);
```

SEE ALSO

xil_get_roi(3), **xil_set_roi**(3), **xil_roi_add_image**(3), **xil_roi_add_rect**(3), **xil_roi_add_region**(3), **xil_roi_create**(3), **xil_roi_create_copy**(3), **xil_roi_destroy**(3), **xil_roi_get_as_image**(3), **xil_roi_intersect**(3), **xil_roi_subtract_rect**(3), **xil_roi_translate**(3), **xil_roi_unite**(3), **XCreateRegion** (3), **XPolygonRegion** (3).

NAME

xil_roi_get_by_name, xil_roi_get_name, xil_roi_set_name – get and set a region of interest (ROI) object name and get a handle to a ROI by specify a name

SYNOPSIS

#include <xil/xil.h>

XilRoi xil_roi_get_by_name (XilSystemState *State***,**
    char ∗*name***);**

char∗ **xil_roi_get_name (XilRoi** *roi***);**

void xil_roi_set_name (XilRoi *roi***,**
    char ∗*name***);**

DESCRIPTION

Use these functions to assign names to ROI objects, get the name of ROIs, and retrieve ROI objects by name.

xil_roi_get_by_name () returns the handle to the ROI object with the specified name *name.* If such an object does not exist, NULL is returned. **xil_roi_get_by_name** () does not make a copy of the ROI object.

xil_roi_get_name () returns a copy of the specified ROI object's name. A call to **free** (3) should be used to free the space allocated by **xil_roi_get_name** (). If the specified ROI object has no name, NULL is returned.

xil_roi_set_name () sets the name of the specified ROI object to the one provided.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Create an ROI named "image1_mask" from an image:

```
XilSystemState State;
XilImage image1;
XilRoi roi;
roi = xil_roi_create(State);
xil_roi_add_image(roi,image1);
xil_roi_set_name(roi, "image1_mask");
```

Use an ROI named "image1_mask" to selectively copy an image:

```
XilSystemState State;
XilImage src, dst;
XilRoi image_mask_roi;
image_mask_roi = xil_roi_get_by_name(State,"image1_mask");
xil_set_roi(dst, image_mask_roi);
xil_copy(src, dst);
```

NOTES

If you give two ROI objects the same name, it is not defined which ROI object will be retrieved by a call to **xil_roi_get_by_name** ().

NAME

xil_roi_intersect – find the intersection of two ROIs

SYNOPSIS

#include <xil/xil.h>

XilRoi xil_roi_intersect (XilRoi *roi1***,**
XilRoi *roi2***);**

DESCRIPTION

This function returns a ROI by taking the intersection of two existing ROIs.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Get the intersection of the ROIs associated with two images:

```
XilImage src, dst;
XilRoi roi_src, roi_dst, roi_intersected;

roi_src = xil_get_roi (src);
roi_dst = xil_get_roi (dst);
roi_intersected = xil_roi_intersect (roi_src, roi_dst);
```

SEE ALSO

xil_get_roi(3), **xil_set_roi**(3), **xil_roi_add_image**(3), **xil_roi_add_rect**(3), **xil_roi_add_region**(3), **xil_roi_create**(3), **xil_roi_create_copy**(3), **xil_roi_destroy**(3), **xil_roi_get_as_image**(3), **xil_roi_get_as_region**(3), **xil_roi_subtract_rect**(3), **xil_roi_translate**(3), **xil_roi_unite**(3).

NAME

xil_roi_subtract_rect – subtract a rectangle from an ROI

SYNOPSIS

#include <xil/xil.h>

void xil_roi_subtract_rect (XilRoi *roi***,**
　　long *x***,**
　　long *y***,**
　　long *width***,**
　　long *height***);**

DESCRIPTION

This function subtracts the specified rectangle from the specified region of interest (ROI). The coordinates of the rectangle are with respect to the storage of the image. That is, an ROI coordinate of (0.0, 0.0) always refers to the upper left pixel in an image, regardless of the image's origin.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Subtract two rectangles from an ROI object, one beginning at (0,0) and ending at (34,94), the other beginning at (10,20) and ending at (109,69):

```
XilRoi roi;
long  xstart, ystart, width, height;

xstart=0; ystart=0; width=35; height=95;
xil_roi_subtract_rect (roi, xstart, ystart, width, height);
xstart=10; ystart=20; width=100; height=50;
xil_roi_subtract_rect (roi, xstart, ystart, width, height);
```

SEE ALSO

xil_get_roi(3), **xil_set_roi**(3), **xil_roi_add_image**(3), **xil_roi_add_rect**(3), **xil_roi_add_region**(3), **xil_roi_create**(3), **xil_roi_create_copy**(3), **xil_roi_destroy**(3), **xil_roi_get_as_image**(3), **xil_roi_get_as_region**(3), **xil_roi_intersect**(3), **xil_roi_translate**(3), **xil_roi_unite**(3).

NAME

xil_roi_translate – translate an ROI up and down or left and right

SYNOPSIS

#include <xil/xil.h>

XilRoi xil_roi_translate (XilRoi *roi***,**
        int *xoffset***,**
        int *yoffset***);**

DESCRIPTION

This function returns a region of interest (ROI) that is translated (moved) (*xoffset, yoffset*) from the specified ROI. The coordinates of the translation are with respect to the storage of the image. That is, an ROI coordinate of (0.0, 0.0) always refers to the upper left pixel in an image, regardless of the image's origin. Positive offsets for *xoffset, yoffset* move the ROI to the right and down. Negative offsets move the ROI left and up.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Move all of the regions comprising an ROI +20 in *x* and -50 in *y* :

```
XilRoi roi, translated_roi;
translated_roi = xil_roi_translate (roi, 20, -50);
```

SEE ALSO

xil_get_roi(3), **xil_set_roi**(3), **xil_roi_add_image**(3), **xil_roi_add_rect**(3), **xil_roi_add_region**(3), **xil_roi_create**(3), **xil_roi_create_copy**(3), **xil_roi_destroy**(3), **xil_roi_get_as_image**(3), **xil_roi_get_as_region**(3), **xil_roi_intersect**(3), **xil_roi_subtract_rect**(3), **xil_roi_unite**(3).

NAME

xil_roi_unite – find the union of two ROIs

SYNOPSIS

#include <xil/xil.h>

XilRoi xil_roi_unite (XilRoi *roi1***,**
    XilRoi *roi2***);**

DESCRIPTION

This function returns a new ROI created by taking the union of two existing ROIs.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Get the union of the ROIs associated with two images:

```
XilImage src, dst;
XilRoi roi_src, roi_dst, roi_union;

roi_src = xil_get_roi (src);
roi_dst = xil_get_roi (dst);
roi_union = xil_roi_unite (roi_src, roi_dst);
```

SEE ALSO

xil_get_roi(3), **xil_set_roi**(3), **xil_roi_add_image**(3), **xil_roi_add_rect**(3), **xil_roi_add_region**(3), **xil_roi_create**(3), **xil_roi_create_copy**(3), **xil_roi_destroy**(3), **xil_roi_get_as_image**(3), **xil_roi_get_as_region**(3), **xil_roi_intersect**(3), **xil_roi_subtract_rect**(3), **xil_roi_translate**(3).

NAME

xil_rotate – rotate an image

SYNOPSIS

#include <xil/xil.h>

void xil_rotate (XilImage *src***,**
XilImage *dst***,**
char **interpolation***,**
float *angle***);**

DESCRIPTION

This function rotates an image about its origin. By default, an image's origin is its upper-left corner (0.0, 0.0). You can change the origin with the **xil_set_origin**() function.

src is the source image handle. *dst* is the destination image handle. *interpolation* is a string that specifies the type of interpolation to be used. The supported interpolation types are *nearest* (nearest neighbor), *bilinear*, *bicubic*, and *general. angle* is the angle of rotation in radians. A positive angle indicates counterclockwise rotation; a negative angle indicates clockwise rotation.

ROI Behavior

If an ROI (region of interest) is attached to the source image, it is used as a read mask and is rotated into the destination image's space, where it is intersected with the destination ROI (if there is one).

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Rotate an image clockwise by 45 degrees (.7854 radians) using bilinear interpolation:

```
XilImage src, dst;
xil_rotate(src, dst, "bilinear", -0.7854);
```

NOTES

The source and destination images to be rotated must be the same type and number of bands. This operation cannot be performed in place.

SEE ALSO

xil_affine(3), **xil_scale**(3), **xil_set_origin**(3), **xil_transpose**(3).

NAME

xil_scale – scale an image

SYNOPSIS

#include <xil/xil.h>

void xil_scale (XilImage *src*,
        XilImage *dst*,
        char ∗*interpolation*,
        float *xscale*,
        float *yscale*);

DESCRIPTION

This function scales an image about its origin. By default, an image's origin is its upper-left corner (0.0, 0.0). You can change the origin with the **xil_set_origin**() function.

src is the source image handle. *dst* is the destination image handle. *interpolation* is a string that specifies the type of interpolation to be used. The supported interpolation types are *nearest* (nearest neighbor), *bilinear, bicubic,* and *general. xscale* and *yscale* are the *x* and *y* scale factors. Scale factors of less than 1.0 reduce the size of an image in *x* and *y*.

ROI Behavior

If an ROI (region of interest) is attached to the source image, it is used as a read mask and is scaled into the destination image's space, where it is intersected with the destination ROI (if there is one).

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Scale an image by 2.3 in the *x* direction and 4.3 in the *y* direction using bicubic interpolation.

```
XilImage src, dst;
xil_scale(src, dst, "bicubic", 2.3, 4.3);
```

NOTES

The source and destination images to be scaled must be the same type and number of bands. This operation cannot be performed in place.

SEE ALSO

xil_affine(3).

NAME

xil_sel_create, xil_sel_create_copy, xil_sel_destroy – create and destroy structuring element objects

SYNOPSIS

#include <xil/xil.h>

XilSel xil_sel_create (XilSystemState *State***,**
    unsigned int *width***,**
    unsigned int *height***,**
    unsigned int *keyx***,**
    unsigned int *keyy***,**
    unsigned int **data***);**

XilSel xil_sel_create_copy (XilSel *sel***);**

void xil_sel_destroy (XilSel *sel***);**

DESCRIPTION

These routines create and control access to the structuring element (SEL) objects used in the XIL erosion and dilation imaging operations. Structuring elements are similar to convolution kernels, except that the member values are Boolean (*unsigned int*).

width and *height* are the width of the structuring element in pixels. Common sizes for structuring elements are 3-by-3 and 5-by-5. *keyx* and *keyy* are the coordinates of the key value in the kernel. The coordinates are specified with respect to the upper-left value in the structuring element (0,0). *data* is a pointer to the Boolean values that will be written to the kernel.

Key values specify the key pixel position - a position relative to the upper left corner of the SEL. The key pixel aligns with the output pixel and constrains which input pixels are used to generate the output.

xil_sel_create () creates a SEL of the specified size with the specified data.

xil_sel_create_copy () returns a copy of the specified SEL. The name of a copy is initially empty (NULL).

xil_sel_destroy () destroys the specified SEL.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Create a 3x3, cross-shaped structuring element, with the key value located at the center of the SEL:

```
XilSystemState State;
unsigned int width=3, height=3, key_x=1, key_y=1;
XilSel sel;
unsigned int data[]={    0, 1, 0,
                         1, 1, 1,
                         0, 1, 0 };

sel = xil_sel_create (State, width, height, key_x, key_y, data);
```

NOTES

The key pixel must lie within the boundaries of the SEL.

SEE ALSO

xil_erode(3), **xil_dilate**(3), **xil_sel_get_height**(3), **xil_sel_get_width**(3), **xil_sel_get_key_x**(3), **xil_sel_get_key_y**(3).

NAME

xil_sel_get_by_name, xil_sel_get_name, xil_sel_set_name – get and set a structuring element (SEL) object name and get a handle to a SEL by specifying its name

SYNOPSIS

```
#include <xil/xil.h>

XilSel xil_sel_get_by_name (XilSystemState State,
        char *name);

char* xil_sel_get_name (XilSel sel);

void xil_sel_set_name (XilSel sel,
        char *name);
```

DESCRIPTION

Use these functions to assign names to SEL objects, get the name of a SEL, and to retrieve SEL objects by name.

xil_sel_get_by_name () returns the handle to the SEL object with the specified name *name.* If such a SEL object does not exist, NULL is returned. **xil_sel_get_by_name** () does not make a copy of the SEL object.

xil_sel_get_name () returns a copy of the specified SEL object's name. A call to **free** (3) should be used to free the space allocated by **xil_sel_get_name** (). If the specified SEL object has no name, NULL is returned.

xil_sel_set_name () sets the name of the specified SEL object to the one provided.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Create a structuring element named "rect3x3":

```
XilSystemState State;
XilSel sel;
unsigned int data[]  =     {1  1  1
                             1  1  1
                             1  1  1 };

sel = xil_sel_create(State,3,3,0,0,data);
xil_sel_set_name(sel, "rect3x3");
```

Use a structuring element named "rect3x3" to erode an image:

```
XilSystemState State;
XilImage src, dst;
XilSel sel;

sel = xil_sel_get_by_name(State,"rect3x3");
xil_erode(src, dst, sel);
```

SEE ALSO **xil_sel_create**(3), **xil_sel_destroy**(3), **xil_sel_get_name**(3), **xil_sel_set_name**(3).

NAME

xil_sel_get_height, xil_sel_get_width, xil_sel_get_key_x, xil_sel_get_key_y – read the values of structuring element attributes

SYNOPSIS

#include <xil/xil.h>

unsigned int xil_sel_get_height (XilSel *sel***);**

unsigned int xil_sel_get_width (XilSel *sel***);**

unsigned int xil_sel_get_key_x (XilSel *sel***);**

unsigned int xil_sel_get_key_y (XilSel *sel***);**

DESCRIPTION

These routines read the values of structuring element (SEL) objects used in erosion and dilation imaging operations. Key values specify the key pixel position - a position relative to the upper left corner of the SEL. The key pixel aligns with the output pixel and constrains which input pixels are used to generate the output.

xil_sel_get_height () gets the height of the specified SEL.

xil_sel_get_width () gets the width of the specified SEL.

xil_sel_get_key_x () gets the *x* coordinate of the key value of the specified SEL.

xil_sel_get_key_y () gets the *y* coordinate of the key value of the specified SEL.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Get the coordinates of a SEL's key value:

```
XilSel sel;
unsigned int key_x, key_y;

key_x = xil_sel_get_key_x (sel);
key_y = xil_sel_get_key_y (sel);
```

SEE ALSO

xil_erode(3), **xil_dilate**(3), **xil_sel_create**(3), **xil_sel_create_copy**(3), **xil_sel_destroy**(3).

NAME

xil_set_colorspace – set an image's color space

#include <xil/xil.h>

void xil_set_colorspace (XilImage *image***,**
  XilColorspace *cspace***);**

DESCRIPTION

This function specifies the *XilColorspace* object associated with the image. The default value of this attribute is *NULL,* which means the image has no color space attached to it.

Images can be supplied in any of the supported color spaces. The following table indicates the character string used to identify the supported color spaces and describes the source of each color space definition:

"rgb709"	Nonlinear RGB primaries as defined by CCIR Rec 709
"rgblinear"	Linearized RGB using primaries from CCIR Rec 709
"ycc709"	YCC as defined by CCIR Rec 709
"y709"	Luminance (black and white) from "ycc709"
"ylinear"	Linearized version of "y709"
"photoycc"	YCC color space defined by Kodak for PhotoCD
"ycc601"	YCC as defined by CCIR Rec 601
"y601"	Luminance from "ycc601"
"cmy"	Linear CMY, derived from "rgblinear"
"cmyk"	Linear CMYK, derived from "cmy" through undercolor removal

These color spaces are created by the XIL library at the time of a call to **xil_open**(3). Handles to these color space objects can be obtained by calling **xil_colorspace_get_by_name**(3).

XIL Color Spaces

The XIL library supports specification of the color spaces of images and the conversion of images between supported color spaces. Color space conversion is useful for a number of reasons.

Some operations are more easily performed on certain color spaces. JPEG compression, for example, produces better results on color data when the input is supplied as YCC instead of RGB. Extracting luminance information from color data allows the simple use of monochrome output devices. The library supports conversion between a variety of these spaces, and treats luminance as a separate color space.

In most cases for 16-bit image data, there is little concern with artifacts due to limited precision. For 8-bit data, using nonlinear or gamma-corrected color spaces (such as YCC or nonlinear RGB) can prevent the contouring in low-intensity regions of the image that occurs with 8-bit linear data storage. The library supports both linear and nonlinear color spaces in both 8 and 16 bits.

Color separations produce images for output on subtractive color printers. The XIL library supports both CMY and CMYK spaces. Some flexibility in the generation of black color (K) and the associated undercolor removal is provided. The library also provides the ability to separate images into a specified group of process colors by dithering to a user-defined colormap. Sophisticated separations (nonlinear black mappings, for example) are not currently supported by the XIL library. Currently, the library only supports certain standard, or objective, color spaces.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

```
XilSystemState State;
XilImage image;
XilColorspace cspace;

/* get handle to the predefined "rgblinear" colorspace */
/* and specify this colorspace for image */
State = xil_open();
image = xil_create(State);
cspace = xil_colorspace_get_by_name(State, "rgblinear");
xil_set_colorspace(image, cspace);
```

SEE ALSO

xil_colorspace_get_by_name(3), **xil_color_convert**(3), **xil_black_generation**(3), **xil_open**(3).

NAME

xil_set_pixel, xil_get_pixel – set or get the value of a particular pixel in an image

SYNOPSIS

#include <xil/xil.h>

```
void xil_set_pixel ( XilImage image,
        unsigned int x,
        unsigned int y,
        float *pixel_values);

void xil_get_pixel ( XilImage image,
        unsigned int x,
        unsigned int y,
        float *pixel_values);
```

DESCRIPTION

xil_set_pixel () sets the value of a particular pixel in an *image. x* and *y* indicate the position of the pixel to be set or read, and *pixel_values* is an array of floats specifying the value to set for each band of the image. Note that the user must allocate and free the space for this array. Pixel coordinates are located with respect to the upper left corner of the image (0,0) whether it is a parent or a child image.

For XIL_BIT images, values below 0.5 cause the pixel to be set to 0, and values 0.5 and above cause the pixel to be set to 1. For XIL_BYTE images, values below 0.5 cause the pixel to be set to 0, values of 254.5 and above cause the pixel to be set to 255, and all values in between are rounded to the nearest integer. For XIL_SHORT images, values below -32768.5 cause the pixel to be set to -32768, values of 32766.5 and above cause the pixel to be set to 32767, and all values in between are rounded to the nearest integer.

xil_get_pixel () gets the value of a particular pixel in an image, and writes a vector of the pixel band values into the user-supplied buffer *pixel_values.* The pixel values are cast from whatever data type they may be into floats.

ROI Behavior

The image ROI is ignored for these operations.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Get the vector of data values out of a 5-banded XIL_BYTE image for the pixel located at (100,42), and add 1.0 to each value:

```
XilImage image;
unsigned int i;
float* pixel_values;

pixel_values = (float *) malloc (5 * sizeof(float));        /* allocate pixel values buffer */
xil_get_pixel (image, 100, 42, pixel_values);               /* get current values of the pixel */
for (i = 0; i < 5; i++)                                     /* increment values by 1 */
   pixel_values[i] = pixel_values[i] + 1.0;
xil_set_pixel (image, 100, 42, pixel_values);               /* replace values in the pixel */
```

SEE ALSO **xil_set_value**(3)

NAME

xil_set_value – set pixels of an image to constant values

SYNOPSIS

#include <xil/xil.h>

void xil_set_value (XilImage *dst***,**
    float **constants***);**

DESCRIPTION

This routine assigns floating point constant values on a pixel-by-pixel basis to the *dst* (destination) image. For an n-band image, the array of *constants* must contain n floating point values. Pixel values are clipped according to image data type.

For XIL_BIT images, values below 0.5 cause the pixel to be set to 0, and values 0.5 and above cause the pixel to be set to 1. For XIL_BYTE images, values below 0.5 cause the pixel to be set to 0, values above 254.5 cause the pixel to be set to 255, and all values in between are rounded to the nearest integer. For XIL_SHORT images, values below -32768.5 cause the pixel to be set to -32768, values above 32766.5 cause the pixel to be set to 32767, and all values in between are rounded to the nearest integer.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Assign pixel values of a 4-band image and store the result in *dst:*

```
XilImage dst;
float values[4];

values[0] = 1.0;
values[1] = 127.5;
values[2] = 256.0;
values[3] = 0.0;

xil_set_value(dst, values);
```

SEE ALSO

xil_set_pixel(3)

NAME

xil_soft_fill – perform soft fill from specified starting point

SYNOPSIS

#include <xil/xil.h>

void xil_soft_fill (XilImage *src,*
    XilImage *dst,*
    float *xseed,*
    float *yseed,*
    float **foregnd_color,*
    unsigned int *num_backgnd_color,*
    float **backgnd_color,*
    float **fill_color);*

DESCRIPTION

This function performs a soft fill on a region composed of the foreground color and a number of background colors. From the starting coordinates, every 4-connected pixel containing a percentage of foreground color is filled with the corresponding percentage of fill color. If a pixel does not contain the foreground color, it forms part of the boundary of the region.

src is the source image handle. *dst* is the destination image handle. *xseed* is a float that specifies the *x* start coordinate. *yseed* is a float that specifies the *y* start coordinate. *foregnd_color* is a pointer to the floating-point array that specifies the foreground color [0...(nbands-1)] for each pixel in the soft fill region.

num_backgnd_color is the number of background colors in the background color list. *backgnd_color* is a pointer to the floating-point array that specifies the list of background colors [num_backgnd_color][0...(nbands-1)] for each pixel in the soft fill region. *fill_color* is a pointer to the floating-point array that specifies the fill color [0...(nbands-1)] for each pixel in the soft fill region.

ROI Behavior

This function performs the fill operation on the entire source image. The filled pixels within the ROI (region of interest) are output to the destination image.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

For this example, the source and destination images contain 3 bands. The foreground color and 2 background colors form the soft fill region. Perform soft fill starting at (x,y) = (7,3).

```
XilImage src;
XilImage dst;
float xseed = 7.0;
float yseed = 3.0;
float foregnd_color[3] = {255.0, 0.0, 0.0};
unsigned int num_backgnd_color = 2;
float backgnd_color[6] = {0.0, 0.0, 0.0, 0.0, 0.0, 255.0};
float fill_color [3] =  {0.0, 255.0, 0.0};

xil_soft_fill(src, dst, xseed, yseed, foregnd_color,
        num_backgnd_color, backgnd_color, fill_color);
```

NOTES

Source and destination images must be the same data type, and have the same number of bands. For an n-band image, the array of floats for *foregnd_color* and *fill_color* must be of size n, and *backgnd_color* must be of size n∗ *num_backgnd_color*.

The set of basis colors, the foreground and background colors, must not be coplanar, or the algorithm will fail to determine the correct percentage for fill color. Only pixels that are changed to the fill color are output to the destination image.

In-place operations are supported.

SEE ALSO

xil_fill(3)

NAME

xil_squeeze_range – produce a lookup table that will map an image into contiguous entries

SYNOPSIS

#include <xil/xil.h>

XilLookup xil_squeeze_range (XilImage *src***);**

DESCRIPTION

This function examines the source image, *src,* and produces a lookup table that will map *src* into an image with contiguous entries. *src* must be a single-banded image. Both *src* and the image's colormap must be passed through the resulting lookup table for it to be displayed correctly.

RETURN VALUES

NULL	if function fails

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Produce a lookup table that will map an image into continuous entries:

```
XilLookup result_lut;
XilImage src;
result_lut = xil_squeeze_range(src);
```

SEE ALSO

xil_lookup(3)

NAME

xil_state_get_interpolation_tables, xil_state_set_interpolation_tables – set or get interpolation tables to or from the XilSystemState object.

SYNOPSIS

#include <xil/xil.h>

void xil_state_get_interpolation_tables (XilSystemState *State***,**
    XilInterpolationTable∗ *horiz***,**
    XilInterpolationTable∗ *vertical***);**

void xil_state_set_interpolation_tables (XilSystemState *State***,**
    XilInterpolationTable *horiz***);**
    XilInterpolationTable *vertical***);**

DESCRIPTION

XIL supports general interpolation. These tables affect all general interpolation operations using images created from this *XilSystemState.* The *horiz* and *vertical* tables define the values in the subsamplng kernels.

xil_state_get_interpolation_tables () gets the interpolation tables of *State.* The *horiz* argument returns a pointer to the horizontal table, and the *vertical* argument returns a pointer to the vertical table. Either table's pointer lets you access that table's kernel size, number of subsamples, and kernel data.

xil_state_set_interpolation_tables () sets the interpolation tables of *State.*

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Set interpolation tables on an XilSystemState object.

```
XilSystemState State;
XilInterpolationTable horiz, vertical;
float ∗horiz_data, ∗vertical_data;

horiz = xil_interpolation_table_create(State, 9, 32, horiz_data);
vertical = xil_interpolation_table_create(State, 9, 32, vertical_data);

xil_state_set_interpolation_tables(State, horiz, vertical);
```

SEE ALSO

xil_interpolation_table_create(3), **xil_interpolation_table_destroy**(3), **xil_interpolation_table_get_subsamples**(3), **xil_interpolation_table_get_kernel_size**(3), **xil_interpolation_table_get_data**(3), **xil_state_get_interpolation_tables**(3).

NAME

xil_state_get_show_action, xil_state_set_show_action – show information about when deferred actions are taken and which actions have been put together into molecules

SYNOPSIS

#include <xil/xil.h>

int xil_state_get_show_action (XilSystemState *State***);**

void xil_state_set_show_action (XilSystemState *State***,**
  int *env_on_off***);**

DESCRIPTION

XIL provides a deferred execution facility that automatically recognizes certain sequences of XIL functions (atoms) and executes the sequences as a single high-performance molecule. An example is a sequence of XIL functions that scales (implicitly capturing) and compresses an image. XIL defers execution of the scale function to see if a compression function follows. If it does, the two functions are executed together as a high-performance molecule. XIL defines a set of general-purpose molecules that perform sequences of operations such as color conversion and decompression.

To determine if XIL functions are executing within molecules, set the SHOW_ACTION attribute of *XilSystemState.* This causes the XIL library to print a message to *stderr* whenever an operation that affects an XIL image or compressed image sequence is executed.

xil_state_get_show_action () gets the current value of the SHOW_ACTION attribute of *State.*

xil_state_set_show_action () sets the current value of the SHOW_ACTION attribute of *State.*

When SHOW_ACTION is set to -1, the XIL library checks the value of the environment variable XIL_DEBUG, and it sets the attribute SHOW_ACTION to 0 if the environment variable XIL_DEBUG does not contain the string "show_action"; it sets the attribute to 1 if XIL_DEBUG contains the string "show_action".

The default value for SHOW_ACTION is -1. When SHOW_ACTION is 1, the library prints information to *stderr* about when deferred actions happen and when they are combined into molecules. When SHOW_ACTION is 0, no information is printed.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Show the output of XIL_SHOW_ACTION in a segment of code, but only if the XIL_DEBUG environment variable is set to "show_action".

```
XilSystemState State;
State = xil_open();
xil_state_set_show_action(State, 0);   /* turn off default behavior */
/* ... set up code ... */
xil_state_set_show_action(State, -1);  /* turn on output (only if environment */
                                       /* variable is set) */
/* ... area of interest ... */
```

```
        xil_state_set_show_action(State, 0);   /* turn off output */
```

NOTES

These functions do not produce any semantic differences in the execution of the program. They are only useful for debugging and performance tuning. Consult the *XIL Programmer's Guide* for information on performance tuning.

NAME

xil_subsample_adaptive – adaptively subsample an image

SYNOPSIS

#include <xil/xil.h>

void xil_subsample_adaptive (XilImage *src***,**
XilImage *dst***,**
float *xscale***,**
float *yscale***);**

DESCRIPTION

This function adaptively subsamples an image about its origin. By default, an image's origin is its upper-left corner (0.0, 0.0). You can change the origin with the **xil_set_origin**() function. The subsampling algorithm used minimizes information loss from skipped pixels in the source image. *src* is the source image handle. *dst* is the destination image handle. *xscale* and *yscale* are the *x* and *y* scale factors, which must be less than or equal to 1.0 and greater than 0.0.

ROI Behavior

If an ROI (region of interest) is attached to the source image, it is used as a read mask and is scaled into the destination image's space, where it is intersected with the destination ROI (if there is one).

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Adaptively subsample an image by .3 in the *x* direction and .4 in the *y* direction:

```
XilImage src, dst;
xil_subsample_adaptive(src, dst, .3, .4);
```

NOTES

This operation cannot be performed in place.

SEE ALSO

xil_subsample_binary_to_gray(3), **xil_set_origin**(3), **xil_scale**(3).

NAME

xil_subsample_binary_to_gray – subsample a binary image and produce a grayscale (byte) image

SYNOPSIS

#include <xil/xil.h>

void xil_subsample_binary_to_gray (XilImage *src***,**
    XilImage *dst***,**
    float *xscale***,**
    float *yscale***);**

DESCRIPTION

This function subsamples a binary image and produces a grayscale (byte) image. *src* is the source image handle. *dst* is the destination image handle. *xscale* and *yscale* are the *x* and *y* scale factors, which must be less than or equal to 1.0 and greater than 0.0.

The subsampling algorithm performs the scaling operation by accumulating all the bits in the source image that correspond to the destination pixel and, based on the *x* and *y* scaling factors, reserving consecutive indexes in the colormap for the maximum number of gray levels possible in the destination image. You must modify your colormap to define a gray level for each resulting index.

For representing the source block of pixels that is used to determine destination pixel values, the index 0 represents a block with no 1's (all 0's), the index 1 represents a block with a single 1, and so on. If the scaling factors require a fractional block of source pixels to determine the destination pixel values, the block size is rounded up. For example, if a 2.2-by-2.2 block of source pixels would be required to determine destination pixel values, a 3-by-3 block is used, resulting in 10 possible gray levels and therefore 10 colormap indexes, whose values are 0 through 9.

ROI Behavior

If an ROI (region of interest) is attached to the source image, it is used as a read mask and is scaled into the destination image's space, where it is intersected with the destination ROI (if there is one).

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Subsample a binary image by .3 in the *x* direction and .4 in the *y* direction to produce a byte image:

```
XilImage src, dst;
xil_subsample_binary_to_gray(src, dst, .3, .4);
```

NOTES

This operation cannot be performed in place.

SEE ALSO

xil_subsample_adaptive(3), **xil_scale**(3).

NAME

xil_subtract, xil_subtract_const, xil_subtract_from_const – image subtraction operations

SYNOPSIS

#include <xil/xil.h>

void xil_subtract (XilImage *src1***,**
 XilImage *src2***,**
 XilImage *dst***);**

void xil_subtract_const (XilImage *src1***,**
 float **constants***,**
 XilImage *dst***);**

void xil_subtract_from_const (float **constants***,**
 XilImage *src1***,**
 XilImage *dst***);**

DESCRIPTION

xil_subtract () performs a pixel-by-pixel subtraction of the *src2* image from the *src1* image and stores the result in the *dst* (destination) image. If the result of the operation is out of range for a particular data type, the result is clamped to the minimum or maximum value for the data type. Results for XIL_BYTE operations, for example, are clamped to 0 if they are less than 0 and 255 if they are greater than 255.

xil_subtract_const () performs a pixel-by-pixel subtraction of the *constants* values from the *src1* image and stores the result in the *dst* (destination) image. For an n-band image, n float values must be provided, one per band. The value in *constants[0]* is subtracted from the values in band 0 of *src* and so on. If the result of the operation is out of range for a particular data type, the result is clamped to the minimum or maximum value for the data type. Results for XIL_BYTE operations, for example, are clamped to 0 if they are less than 0 and 255 if they are greater than 255.

xil_subtract_from_const () performs a pixel-by-pixel subtraction of the *src1* image from the *constants* values and stores the result in the *dst* (destination) image. For an n-band image, n float values must be provided, one per band. The values in band 0 of *src* are subtracted from the value in *constants[0]* and so on. Resulting pixel values are rounded and clipped according to image data type.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Subtract *image2* from *image1* and store the result in *dst* :

```
XilImage image1, image2, dst;

xil_subtract(image1, image2, dst);
```

Subtract *constants* values from 4-band *image1* and store the result in *dst* :

```
XilImage image1, dst;
float constants[4];
```

```
constants[0] = 1.0;
constants[1] = 1.0;
constants[2] = 1.0;
constants[3] = 0.0;
xil_subtract_const(image1, constants, dst);
```

Subtract 4-band *image1* from *constants* values and store the result in *dst* :

```
XilImage image1, dst;
float constants[4];

constants[0] = 255.0;
constants[1] = 255.0;
constants[2] = 255.0;
constants[3] = 255.0;
xil_subtract_from_const(constants, image1, dst);
```

NOTES

Source and destination images must be the same data type and have the same number of bands. In-place operations are supported.

SEE ALSO

xil_add(3), **xil_add_const**(3).

NAME

xil_sync, xil_get_synchronize, xil_set_synchronize, xil_state_get_synchronize, xil_state_set_synchronize – force computation of the value of an image when it would otherwise have been deferred

SYNOPSIS

#include <xil/xil.h>

void xil_sync (XilImage *image***);**

Xil_boolean xil_get_synchronize (XilImage *image***);**

void xil_set_synchronize (XilImage *image***,**
    Xil_boolean *onoff***);**

Xil_boolean xil_state_get_synchronize (XilSystemState *State***);**

void xil_state_set_synchronize (XilSystemState *State***,**
    Xil_boolean *onoff***);**

DESCRIPTION

xil_sync () forces the computation of the value of an image in cases in which that operation might otherwise have been deferred. This prevents deferred execution from attempting to optimize beyond the point at which the **xil_sync** (3) call is made.

xil_get_synchronize () and **xil_set_synchronize** () set and get the synchronization status of an image. If an image is synchronous, operations on that image are never deferred.

xil_state_get_synchronize () and **xil_state_set_synchronize** () turn synchronization on or off for *State.* The default synchronization for *State* is FALSE, which means that deferred execution is used. If the synchronization status of *State* is set to TRUE, then all operations are executed immediately -- no deferral occurs.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Measure the performance of an image rotate operation with bilinear interpolation:

```
starttime= timelocal(NULL);              /* get the start time */
xil_rotate(src,dst,"bilinear",0.5);/* make dst a rotation of source */
xil_sync(dst);                           /* force the rotate to actually happen */
endtime= timelocal(NULL);                /* get the finish time */
```

NOTES

None of these functions produces any semantic differences in the execution of the program. These functions are only useful for debugging, performance measurement, and performance tuning.

SEE ALSO

xil_cis_sync(3)

NAME

xil_tablewarp, xil_tablewarp_horizontal, xil_tablewarp_vertical – warp an image with a user-specified warp table

SYNOPSIS

#include <xil/xil.h>

void xil_tablewarp (XilImage *src***,**
        XilImage *dst***,**
        char* *interpolation***,**
        XilImage *warp_table***);**
void xil_tablewarp_horizontal (XilImage *src***,**
        XilImage *dst***,**
        char* *interpolation***,**
        XilImage *warp_table***);**

void xil_tablewarp_vertical (XilImage *src***,**
        XilImage *dst***,**
        char* *interpolation***,**
        XilImage *warp_table***);**

DESCRIPTION

These functions warp an image with the specified warp table. *src* is the source image handle. *dst* is the destination image handle. *interpolation* is a string that specifies the interpolation to be used. The supported interpolation types are nearest (nearest neighbor), bilinear, bicubic, and general. *warp_table* is a handle to an *XilImage* structure that describes the backward mapping from a pixel in the destination to a pixel in the source.

A warp table is an XIL image whose pixel values define the backward mapping from a pixel in the destination to a pixel in the source. The warp table is applied at the origin of the destination image. The source origin is then added to the backward mapping position specified by the warp table. A warp table must have the datatype XIL_SHORT, though it can be used to warp images of any data type. The XIL_SHORT value is interpreted as fixed point wth 12 bits value and 4 bits of precision.

The warp table for **xil_tablewarp** () is a 2-banded image where the bands specify the displacement in *x* and the displacement in *y*. The warp table for **xil_tablewarp_horizontal** () and **xil_tablewarp_vertical** () is 1-banded and specifies the displacement in the *x* and *y* directions, respectively.

ROI Behavior

Because a warp table is technically an XIL image, it can have a defined region of interest (ROI). However, an ROI is meaningless in a warp table and is therefore ignored.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

For this example, a warp table is created to produce the same effect as translation. This example translates a 100 x 120 block from src origin to the right and down with offset (26.0, 37.0) using bilinear interpolation.

```
XilSystemState State;
XilImage src, dst, warp_table;
float values[2];

warp_table = xil_create(State, 100, 120, 2, XIL_SHORT);
/* multiply offsets by 16 because of 12 bit values with 4 bit precision */
values[0] = 26.0 * 16;
values[1] = 37.0 * 16;
xil_set_value(warp_table, values);

xil_tablewarp(src, dst, "bilinear", warp_table);
```

NOTES

Source and destination images must be the same data type and have the same number of bands. The images need not have the same width and height. This operation cannot be performed in place.

SEE ALSO

xil_set_origin(3), **xil_set_pixel**(3), **xil_set_value**(3).

NAME

xil_threshold – set value of image pixel bands within a specified range

SYNOPSIS

#include <xil/xil.h>

void xil_threshold (XilImage *src*,
  XilImage *dst*,
  float ∗*low*,
  float ∗*high*,
  float ∗*map*);

DESCRIPTION

For each band of an image, this function maps to a constant all the values that fall between a low value and a high value. *src* is the source image handle. *dst* is the destination image handle. *low* is a pointer to the floating-point array that specifies the low value of the range for band [0...(nbands-1)]. *low[0]* is the low value for band 0, and so forth. *high* is a pointer to the floating-point array that specifies the high value of the range for band [0...(nbands-1)]. *high[0]* is the high value for band 0, and so forth. *map* is a pointer to the floating-point array that specifies the map value for each pixel band within the range [low:high].

For an n-band image, the array of floats for *low, high,* and *map* must be of size n. Each band is independently evaluated for its range. Values outside the range are passed through without change.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

For this example, the source and destination images contain 2 bands. Force each pixel in band[0] between [192:255] to value 191. Force each pixel in band[1] between [0:63] to value 64.

```
XilImage src;
XilImage dst;
float low[2] =  {192.0, 0.0};
float high[2] = {255.0, 63.0};
float map[2] =  {191.0, 64.0};
xil_threshold(src, dst, low, high, map);
```

NOTES

Source and destination images must be the same data type and have the same number of bands. In-place operations are supported.

NAME

xil_to_xgl, xgl_to_xil – convert between XIL images and XGL rasters

SYNOPSIS

#include <xil/xil.h>
#include <xgl/xgl.h>

Xgl_ras xil_to_xgl (XilImage *src***,**
    Xgl_sys_state *xgl_state***);**

XilImage xgl_to_xil (Xgl_ras *src***,**
    XilSystemState *xil_state***);**

DESCRIPTION

These functions convert images between the XIL library's memory / display images and the XGL library's memory / display rasters. *src* is the source image / raster handle. If the function is successful, a handle to a raster / image is returned. Note that it is not possible to convert an XGL memory raster if an XIL memory image's scanlines are not 16-bit aligned, because of the alignment difference between XIL memory images and XGL memory rasters. Supported conversions are as follows:

XGL_COLOR_INDEX display raster	<->	XIL one-band byte display image
XGL_COLOR_RGB display raster	<->	XIL three-band byte display image
XGL_COLOR_INDEX memory raster	<->	XIL one-band byte memory image
XGL_COLOR_RGB memory raster	<->	XIL four-band byte memory image

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

```
XilSystemState State;
Xgl_sys_state xgl_state;
Display *display;
Window window;
XilImage memory_image, display_image;
Xgl_ras win_ras, mem_ras;
Xgl_ctx ctx;

xil_state = xil_open();
xgl_state = xgl_open(NULL);

display_image = xil_create_from_window(State, display, window);
memory_image = xil_create(State, 512, 512, 4, XIL_BYTE);

win_ras = xil_to_xgl(display_image, xgl_state);
mem_ras = xil_to_xgl(memory_image, xgl_state);
ctx = xgl_object_create(xgl_state, XGL_2D_CTX, NULL, XGL_CTX_DEVICE,
        mem_ras, NULL);
```

(XGL library calls)

xgl_context_flush(ctx, XGL_FLUSH_SYNCHRONIZE);

(XIL library calls)

xil_sync(memory_image);

NOTES

An application should call **xgl_context_flush** (*ctx*, *XGL_FLUSH_SYNCHRONIZE*) before switching from the XGL to the XIL library, and call **xil_sync** (*image*) before switching from the XIL to the XGL library. Applications need to link with the following libraries to use these routines: XIL, XGL, and libxil_to_xgl.

This interface to XIL / XGL interoperability is "uncommitted," which means it may change or be replaced by another mechanism in a future release.

SEE ALSO

xgl_object_create (3), **xil_create**(3), **xil_create_from_device**(3), **xil_create_from_window**(3), **xil_sync**(3).

NAME

xil_toss – throw away the contents of an image without destroying it

SYNOPSIS

#include <xil/xil.h>

void xil_toss (XilImage *image***);**

DESCRIPTION

This function throws away the contents of an image without destroying it. This function provides a way to inform the XIL library that the user is no longer concerned about the contents of an image. After **xil_toss**(3) is called, the value of the image is undefined.

This function can sometimes be useful for code optimization. Sometimes the XIL library will perform more optimally if **xil_toss** () is called when the results of an intermediate operation are no longer needed. When used properly, this function will not change the results of operations.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

NOTES

It is illegal to use a tossed image as a source without first using it as a destination.

NAME

xil_translate – translate an image

SYNOPSIS

#include <xil/xil.h>

void xil_translate (XilImage *src***,**
　　XilImage *dst***,**
　　char *∗interpolation***,**
　　float *xoffset***,**
　　float *yoffset***);**

DESCRIPTION

This function translates an image. *src* is the source image handle. *dst* is the destination image handle. *interpolation* is a string that specifies the type of interpolation to be used. The supported interpolation types are *nearest* (nearest neighbor), *bilinear*, *bicubic*, and *general*. *xoffset* and *yoffset* are the number of pixels to translate or shift the image in the horizontal or vertical directions, respectively. Postive values for *xoffset* and *yoffset* shift an image to the right and down, respectively. Negative values shift to left and up.

ROI Behavior

If an ROI (region of interest) is attached to the source image, it is used as a read mask and is translated into the destination image's space, where it is intersected with the destination ROI (if there is one).

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Translate an image by 12.3 pixels horizontally and 43.2 pixels vertically using nearest neighbor interpolation.

```
XilImage src, dst;
xil_translate(src, dst, "nearest", 12.3, 43.2);
```

NOTES

The source and destination images must be the same data type and number of bands. This operation cannot be performed in place.

SEE ALSO

xil_affine(3)

NAME

xil_transpose – rotate or transpose an image

SYNOPSIS

#include <xil/xil.h>

void xil_transpose (XilImage *src***,**
XilImage *dst***,**
XilFlipType *fliptype***);**

DESCRIPTION

This function reflects an image in some direction or rotates an image in multiples of 90 degrees. *src* is the source image handle. *dst* is the 1.1 destination image handle. *fliptype* is an enumeration constant that represents the direction of reflection as follows:

fliptype	*Reflection Direction*
XIL_FLIP_Y_AXIS	horizontal, across the *y* axis
XIL_FLIP_X_AXIS	vertical, across the *x* axis
XIL_FLIP_MAIN_DIAGONAL	transpose across the main diagonal
XIL_FLIP_ANTIDIAGONAL	transpose across the anti-diagonal
XIL_FLIP_90	rotate counterclockwise 90 degrees
XIL_FLIP_180	rotate counterclockwise 180 degrees
XIL_FLIP_270	rotate counterclockwise 270 degrees

ROI Behavior

If an ROI (region of interest) is attached to the source image, it is used as a read mask and is also "flipped" into the destination image’s space, where it is intersected with the destination ROI (if there is one).

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer’s Guide.*

EXAMPLES

Reflect an image vertically across the X axis:

```
XilImage src, dst;
xil_transpose(src, dst, XIL_FLIP_X_AXIS);
```

NOTES

Source and destination images must be the same data type and number of bands. This operation cannot be performed in place. This operation ignores the location of an image’s origin.

NAME

xil_xor, xil_xor_const – bitwise logical XOR operations

SYNOPSIS

```
#include <xil/xil.h>

void xil_xor (XilImage src1,
        XilImage src2,
        XilImage dst);

void xil_xor_const (XilImage src1,
        unsigned int *constants,
        XilImage dst);
```

DESCRIPTION

xil_xor () performs a bitwise logical XOR operation on each pixel of the *src2* (source) image with the *src1* image and stores the result in the *dst* (destination) image.

xil_xor_const () performs a bitwise logical XOR operation on each pixel of the *src1* (source) image with a specified constant and stores the results in the *dst* (destination) image. For a n-band image, n unsigned integers must be provided, one per band. The value of *constants[0]* is XORed with the values in band 0, and so on.

ERRORS

For a complete list of XIL error messages by number, consult Appendix B of the *XIL Programmer's Guide.*

EXAMPLES

Bitwise logical XOR *image2* with *image1* and store the result in *dst:*

```
XilImage image1, image2, dst;

xil_xor(image1, image2, dst);
```

Bitwise logical XOR a 4-band *image1* with 4 constants and store the result in *dst:*

```
XilImage image, dst;
unsigned int constants[4];

constants[0] = 1;
constants[1] = 0;
constants[2] = 0;
constants[3] = 0;
xil_xor_const(image, constants, dst);
```

NOTES

Source and destination images must be the same data type and have the same number of bands. In-place operations are supported.

Index