NAME

tclgd – modern, nearly feature-complete interface to gd-2 graphics drawing routines

SYNOPSIS

package require gdtcl

GD create *objName width height*
GD create_truecolor *objName width height*
GD create_from_jpeg *objName channel*
GD create_from_jpeg_data *objName data*
GD create_from_png *objName channel*
GD create_from_png_data *objName data*
GD create_from_gif *objName channel*
GD create_from_gif_data *objName data*
GD create_from_gd *objName channel*
GD create_from_gd_data *objName data*
GD create_from_gd2 *objName channel*
GD create_from_gd2_data *objName data*
GD create_from_gd2_part *objName channel x y width height*
GD create_from_gd2_part_data *objName data x y width height*
GD create_from_wbmp *objName channel*
GD create_from_wbmp_data *objName data*
GD create_from_xbm *objName fileHandle*
GD create_from_xpm *objName fileHandle*

objName pixel *x y ?color?*
objName pixelrgb *x y ?color?*
objName line *x1 y1 x2 y2 color*
objName polygon *?filled|open? pointList color*
objName rectangle *x1 y1 x2 y2 color*
objName filled_rectangle *x1 y1 x2 y2 color*
objName arc *cx cy width height startDegrees endDegrees color*
objName filled_arc *cx cy width height startDegrees endDegrees color ?arc? ?chord? ?pie? ?nofill? ?edged?*
objName filled_ellipse *cx cy width height color*
objName fill_to_border *x y borderColor color*
objName fill *x y color*
objName fill *x y color*
objName text *color font pointSize angle x y text*
objName text_bounds *color font pointSize angle x y text*
objName allocate_color *r g b ?alpha?*
objName closest_color *r g b ?alpha?*
objName closest_color_hwb *r g b*
objName exact_color *r g b ?alpha?*
objName resolve_color *r g b ?alpha?*
objName total_colors
objName deallocate_color *color*
objName true_color *r g b ?alpha?*
objName interlace *?boolean?*
objName transparent *?color?*
objName set_anti_aliased *color*
objName set_anti_aliased_dont_blend *color*
objName set_brush *brushImageCommand*
objName set_tile *tileImageCommand*
objName set_style *colorList*
objName set_thickness *thickness*

objName alpha_blending *blending*
objName save_alpha *boolean*
objName get_alpha
objName bounds_safe *x y*
objName green_component *color*
objName red_component *color*
objName blue_component *color*
objName rgb_components *color*
objName width
objName height
objName copy *srcImageCommand destX destY srcX srcY width height*
objName copy_resized *srcImageCommand destX destY srcX srcY destWidth destHeight srcWidth srcHeight*
objName copy_resampled *srcImageCommand destX destY srcX srcY destWidth destHeight srcWidth srcHeight*
objName copy_rotated *srcImageCommand destX destY srcX srcY destWidth destHeight srcWidth srcHeight angle*
objName copy_merge *srcImageCommand destX destY srcX srcY width height percent*
objName copy_merge_grey *srcImageCommand destX destY srcX srcY width height percent*
objName copy_palette *srcImageCommand*
objName sharpen *percent*
objName compare *otherImageCommand*
objName compare_ratio *otherImageCommand*
objName square_to_circle *name radius*
objName write_jpeg *channel quality*
objName jpeg_data *quality*
objName write_gif *channel*
objName gif_data
objName gif_anim_begin *channel global_color_map loops*
objName gif_anim_add *channel local_color_map left_offset top_offset delay disposal ?previous_image?*
objName gif_anim_end *channel*
objName write_png *channel compressionLevel*
objName png_data *compressionLevel*
objName write_wbmp *channel fgcolor*
objName wbmp_data *fgcolor*
objName write_gd *fileHandle*
objName gd_data
objName write_gd2 *channel chunkSize format*
objName gd2_data *chunkSize format*

DESCRIPTION

tclgd gives Tcl programs the ability to read, manipulate, and generate graphic images in a number of popular formats.

The **GD** command creates a new Tcl command, either prepping it with an empty graphic for the case of **create** and **create_truecolor**, or by reading file data either from a Tcl *channel*, from in-memory data, or from a Tcl file handle (in a couple of cases where a channel interface can't be provided due to limitations of the underlying gd library routines).

IMPORTANT: The channel being read should be configured for binary translation using something like

```
set fp [open parrots.png]
fconfigure $fp -translation binary
GD create_from_png parrots $fp
```

If you get corrupt image errors on images that you feel pretty sure are OK,
like you can load them into Photoshop, you probably are not setting
translation to binary.

For formats where channels are support (the majority), any type of channel can be used, including files, sockets, and, if so equipped, alternate channel interfaces such as reading the data directly from zip files.

Tweezer gives Tcl programs the ability to examine and manipulate the state of Tcl objects, the core building blocks of Tcl in versions 8 and above.

Once an image has been created or loaded, it can be manipulated via the named object, or if the object name is **#auto**, a unique command name is returned and should be grabbed into a variable and executed using a reference to that variable, in the same manner as Incr Tcl.

```
GD create img 100 100
img pixel 5 5
```

DRAWING FUNCTIONS

objName pixel *x y* returns the color index of the pixel located at coordinates x and y, while **objName pixel** *x y color* sets that pixel location to the specified color index.

objName pixelrgb *x y* returns a list containing the red, green and blue values of the pixel located at coordinates x and y, while **objName pixelrgb** *x y color* sets that pixel location to the specified color index. For setting, use **pixel** instead as how or whether **pixelrgb** will set colors is likely to change.

objName line *x1 y1 x2 y2 color* draws a line from the coordinates *x1, y1* to *x2, y2* using color *color*. If thickness is set to 1 and antialiasing is enabled, the line will be drawn antialiased.

objName polygon *?filled|open? pointList color* draws a polygon of color *color* using points from the point list *pointList* which must contain an even number of xy pairs.

objName rectangle *x1 y1 x2 y2 color* draws a rectangle of the given color from the corner at x1, y2 to the corner at x2, y2.

objName filled_rectangle *x1 y1 x2 y2 color* same as rectangle except the rectangle is filled.

objName arc *cx cy width height startDegrees endDegrees color* draws an arc centered on cx, cy, of the specified width and heigtht, starting and ending at the specified degrees, and using the specified color.

objName filled_arc *cx cy width height startDegrees endDegrees color ?arc? ?chord? ?pie? ?nofill? ?edged?* draws an arc centered on cx, cy, of the specified width and heigtht, starting and ending at the specified degrees, and using the specified color. Additional options can be specified and are logically or'ed together, **chord**, **pie**, **nofill** and/or **edged**.

objName filled_ellipse *cx cy width height color* draws a filled ellipse centered at cx, cy, of the given width, height and color.

objName fill_to_border *x y borderColor color* fills to the border matching the specified borderColor, starting at x, y, using the specified color.

objName fill *x y color* does a flood fill starting at x, y using the specified color.

objName text *color font pointSize angle x y text* renders text using the specified font and point size, at the specified angle, starting at x, y. **Font** is the full or relative pathname to a TrueType font file (.ttf or .ttc file). (**Tclgd** does not support GD's optionally built-in, non-antialiased, non-TrueType fonts.)

objName text_bounds *color font pointSize angle x y text*determines the bounds of text given the specified font and point size, at the specified angle, starting at x, y. The bounds are returned as an 8-element list giving the x and y coordinates of the four corners of a box the text will be within.

objName allocate_color *r g b ?alpha?* allocates a color given the specified red, green and blue values, and optional alpha value, and returns the color index. For truecolor images it returns a value that can be used to draw the specified color, but it's not really an index per se.

objName closest_color *r g b ?alpha?* returns the closest color index that can be found among the currently allocated colors.

objName closest_color_hwb *r g b*

objName exact_color *r g b ?alpha?*

objName resolve_color *r g b ?alpha?*

objName total_colors

objName deallocate_color *color*

objName true_color *r g b ?alpha?*

objName interlace *?boolean?* if set to 1, t, etc, causes the image to be saved interlaced if the output format supports it, if 0, f, etc, causes the image to be saved noninterlaced. If boolean isn't specified, returns the current interlace setting for the image.

objName transparent *color* says what color index will be generated as transparent when the image is saved, assuming the outputted image format supports it. If *color* is not specified, the current color is returned. **-1** means transparency is disabled; setting transparency to -1 disables it as well.

objName set_anti_aliased *color*

objName set_anti_aliased_dont_blend *color*

objName set_brush *brushImageCommand* - set the brush for brushed drawing to be the image contained in the specified image command.

objName set_tile *tileImageCommand* - set the tile for tiled drawing to be the image contained in the specified image command.

objName set_style *colorList* - set the image style for styled lines to a list of colors. Each color in the list is either a color value or the word *transparent* to indicate that the existing color should be left unchanged for that particular pixel, allowing lines to be drawn with dashed lines, etc, when the color argument to the line drawing function is specified as *styled*.

Styles and brushes can be combined to draw the brush image at intervals instead of with a continuous stroke. When creating a style list for use with a brush, list elements of zero indicate pixels at which the brush should not be drawn and elements of one indicates pixels at which the brush should be drawn.

To draw a styled, brushed line, use the special color *styled_brushed* for the color argument to the **draw** method..

objName set_thickness *thickness* - set the thickness of lines drawn by the line and polygon drawing functions.

objName alpha_blending *blending* - GD has two different modes for drawing on truecolor images. In blending mode, which is on by default, the alpha channel component of the color supplied to each drawing function is used to determine how much of the underlying color should be allowed to shine through. In this mode gd automatically blends the existing color at that point with the drawing color, storing the result in the image. In this mode, the resulting pixel is opaque (alpha channel value of zero.)

In non-blending mode, the drawing color is copied literally with its alpha channel information, replacing the destination pixel with the color and alpha channel value. In this mode the alpha value stored in the image is whatever alpha value the pixel was drawn with, hence the resulting pixel in the resulting image will range from opaque to translucent depending on the alpha channel value.

Blending mode is only available with truecolor images and PNG is currently the only file format supported by gd which can include alpha channel information.

objName save_alpha *boolean* - by default gd does not attempt to save full alpha channel information (as opposed to single-color transparency) when saving PNG images. (PNG is currently the only gd-supported file format that can include alpha channel information.) This saves space in the output file.

If you want to create an image with a full alpha (transparency) channel, invoke the **save_alpha** method with an argument of 1 and also invoke the **alpha_blending** method with an argument of 0 to turn off alpa blending within the library, causing the alpha channel information to be stored in the image rather than having gd

composite the image immediately when the drawing functions are invoked.

objName get_alpha *color* - returns the alpha component of the specified color index, where 0 is completely opaque (no blending with the background) through 127 being completely transparent (the background shines through 100%).

objName bounds_safe *x y*

objName bounds_safe *x y*

objName green_component *color* - returns the green component of the specific color index.

objName red_component *color* - returns the red component of the specified colofr index.

objName blue_component *color* - returns the blue component of the specified colofr index.

objName rgb_components *color* - returns the red, green and blue components of the color as a list.

objName width *return the width of the image, while* **objName height** returns the height.

objName compare *otherImageCommand* - given two images (the named object and another image command), return a list of ways in which the two images are different.

The members of the list will include zero or more of the following elements: **image**, if present, indicates that the images will appear differently when displayed. **num_colors**, if present, indicates that the number of colors in the palettes differ, while **colors** indicates that the colors differ between the two images. **height**, if present, indicates that the height of the images differ, while **width**, if present, indicates that the width of the images differs.

transparent, if present, indicates that the transparent color differs, while **background** indicates that the background color differs. **interlace**, if present, indicates that one image is interlaced while the other is not, and **truecolor**, if present, indicates that one image is a truecolor while the other is indexed and has a palette.

If the list returned is empty, the images should be identical. If **image** is not present, the images will appear identically if displayed, regardless of other characteristics differing. According to the libgd documentation, any difference in the transparent color is assumed to make images display differently, even if the transparent color is not used.

objName compare_ratio *otherImageCommand* - given two images of identical dimensions, returns the ratio of pixels that are identical between the two images, between 0.0 and 1.0.

objName copy *srcImageCommand destX destY srcX srcY width height*

objName copy_resized *srcImageCommand destX destY srcX srcY destWidth destHeight srcWidth srcHeight*

objName copy_resampled *srcImageCommand destX destY srcX srcY destWidth destHeight srcWidth srcHeight*

objName copy_rotated *srcImageCommand destX destY srcX srcY destWidth destHeight srcWidth srcHeight angle*

objName copy_merge *srcImageCommand destX destY srcX srcY width height percent*

objName copy_merge_grey *srcImageCommand destX destY srcX srcY width height percent*

objName copy_palette *srcImageCommand*

objName sharpen *percent*

objName square_to_circle *name radius*

IMAGE OUTPUT ROUTINES

objName write_jpeg *channel quality* writes the image in JPEG format, with the specified quality level, to the specified Tcl channel, while **objName jpeg_data** *quality* will return it as binary data. Quality can range from **0** to **100**, where higher numbers give higher quality.

objName write_gif *channel* writes the image to the specified channel in GIF format, while **objName**

gif_data returns it as binary data.

objName write_png *channel compressionLevel* write the image to the specified channel, with the compression level of **-1** for the default set when zlib was built, **0** to indicate no compression, **1** to compress as quickly as possible, all the way to **9** to select the best possible compression. Likewise **objName png_data** *compressionLevel* returns the image as binary data, with the specified compression.

objName write_wbmp *channel fgcolor* write the image to the specified channel in Windows bitmap format, selecting only pixels matching the specified foreground color. **objName wbmp_data** *fgcolor* returns the image in wbmp format as binary data.

objName write_gd *fileHandle* writes the image out in the high-performance but non-portable *gd* format, while **objName gd_data** returns the image as binary data.

objName write_gd2 *channel chunkSize format*

objName gd2_data *chunkSize format* writes the image out in the high-performance but non-portable *gd2* format, with *chunkSize* determining the size of each chunk, and format being either **compressed** or **raw**.

ANIMATED GIFS

objName gif_anim_begin *channel global_color_map loops* begins an animated GIF by specifying the file channel to be written to, a **1** if the global color map is to be used, and the number of times to play the animation. If *loops* is zero, the animation will repeat indefinitely.

objName gif_anim_add *channel local_color_map left_offset top_offset delay disposal ?previous_image?* adds a GIF image to an animated GIF that is under construction. Set *local_color_map* to 1 to add a local palette for this image to the animation, else the global palette is used.
If you use local palettes, you must make sure they match the global palette; use the *copy_palette* method to copy palettes between images.
left_offset and *top_offset* let you place the frame with an offset into the parent frame, where (0, 0) puts it in the corner.
Delay specifies the delay between the previous frame and this frame in hundredths of a second. Disposal can be 0 for unknown, 1 for none, 2 for restore background, or 3 for restore previous. Unknown is not recommended. Restore background restores the first allocated color of the global palette. Restore previous restores the appearance of the affected area before teh frame was rendered. Only /fBnone is a sensible choice for the first frame.
If the previous image is passed, the built-in GIF optimizer is automatically engaged and the disposal method specified is ignored. The optimizer comapres the images and only writes the changed pixels to the new frame of the animation. To achieve good optimization, it is usually best to use a single global color map. To allow *gif_anim|add* to compress unchanged pixels via the use of a transparent color, the image must include a transparent color. See **gd** documentation for details.
objName gif_anim_end *channel* is self-explanatory.