

**NAME**

*pilights* – tcl interface to do fancy things with LED vectors.

**SYNOPSIS**

**package require pilights**

**pilights create** *objName nLights nRows*

*objName* **nLights**

*objName* **nRows**

*objName* **fillrows** *firstRow lastRow r g b*

*objName* **fillpixels** *row firstPixel nPixels r g b*

*objName* **write** *spiObject row ?nRows? ?delay?*

*objName* **copyrows** *firstRow destRow nRows*

*objName* **copy\_from\_image** *gdObject x y firstRow firstPixel nPixels*

*objName* **getrow** *row*

*objName* **setrow** *row list*

**DESCRIPTION**

*pilights* gives Tcl programs the ability to generate sequences for RGB lights and emit them over the Raspberry Pi's SPI bus to stuff like LPD8806 digital RGB LED strips.

The **pilights create** command creates a new Tcl object, prepping it with arrays of bytes for setting up and sending RGB values to many lights very quickly. Latch bytes are preallocated and configured in-place for convenience runtime performance. The rows are initialized to black or "off", values of zero for all red, green and blue pixels.

A cool feature, *pilights* can take pixels from any image the GD library can handle, including PNG, GIF and JPEG, and emit them to the LED strips.

Once an object has been created, it can be manipulated via its name, 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.

```
pilights create mylights 150 100
```

```
mylights fill 0 100 0 0 255
```

**PILIGHTS OBJECT INTROSPECTION FUNCTIONS**

*objName* **nLights** returns the number of lights defined, while *objName* **nRows** returns the number of rows defined.

*objName* **getrow** *row* returns the pixels of the specified as a flat list of red, green and blue values. If there are 150 lights defined, for instance, a list of 450 integers is returned.

**PIXEL-SETTING FUNCTIONS**

*objName* **fillrows** *firstrow lastRow r g b* fills the specified rows with the specified RGB colors. Use 8-bit colors. They are mapped to 7-bit for emission to the string but we treat the interface as if it is 8-bit. For example, 255 255 255 is full brightness even though the LED gets values of 127 127 127.

*objName* **fillpixels** *row firstPixel nPixels r g b* fills pixels within a row with RGB colors. Use it multiple times with different RGB components, first pixel and number of pixels to create patterns within a row.

*objName* **copyrows** *firstRow destRow nRows* copies patterns one row to another.

*fBobjName* **copy\_from\_image** *gdObject x y firstRow firstPixel nPixels* reads pixels from a GD library image (probably loaded from a PNG or GIF) using the *tcl.gd* package. This allows pixel sequences of nearly unlimited size and complexity to be created and stored efficiently within PNG24 images.

Pixels are copied pixel-for-pixel. If the number of pixels requested runs off the end of one row in the *pilights* object, copying continues with the first pixel of the next row. Likewise if the row of pixels in the GD image runs out, pixels continued to be sourced from the Xth pixel in the next row.

*objName* **setrow** *row list* sets the pixels of the specified row from a flat list of red, green and blue values. If the list contains fewer than the needed number of values to contain a red, green and blue value for each

pixel, it is repeated circularly until all the pixels are set. If there are three values in the list, for instance, each pixel will be set to the same red, green and blue values. If there are six values, the pixels will alternate in pairs.

For sanity, the list must contain a minimum of three elements and its length must be a multiple of three.

Note that since the RGB values are stored lamp-ready and retrieved from that state, the least significant bit of the eight-bit color that was originally set is lost. A pixel value of one is returned as zero, a pixel value of two or three is returned as two, and so on.

## WRITING TO LEDs

To write to LEDs using the SPI bus, you also need the `tcspi` package. Set up a `tcspi` object for talking to the SPI bus by doing a *package require spi* and *set spi [spi #auto /dev/spidev0.0]*.

Write a row of pixels to the SPI bus using *objName write spiObject row ?nRows? ?delay?*. *spiObject* is a result of a successful creation of an SPI object by `tcspi`. *row* is the row number. *nRows*, if present, specifies the number of rows to write, iterating sequentially through the rows. If absent, only a single row is emitted.

*Delay* is the delay in microseconds, an argument to the `ioctl` that `tcspi` uses to talk to the SPI hardware.

More to come...