

Thorlabs Mono To Color Processing C API Reference

Rev. 2.0 2021-08-05 ITN004107-D01

Contents

0.1	Thorlabs Scientific Mono To Color Processor	1
0.1.1	Introduction	1
0.1.2	Getting Started	1
0.1.3	Example	4
0.1.4	Advanced Color Processing	12
0.2	File Index	12
0.2.1	File List	12
0.3	File Documentation	12
0.3.1	tl_mono_to_color_enum.h File Reference	12
0.3.1.1	Enumeration Type Documentation	13
0.3.2	tl_mono_to_color_processing.h File Reference	15
0.3.2.1	Typedef Documentation	16
Index	31

0.1 Thorlabs Scientific Mono To Color Processor

0.1.1 Introduction

The target audience for this document are software engineers with knowledge of C and Thorlabs cameras. Knowledge of color processing is helpful but not required.

The mono to color processing sdk is an all-in-one module to easily transform monochrome image data into colored image data. It wraps around the color processing suite, which is a more complex set of libraries that allows one to control every aspect of color processing. See the [Advanced Color Processing](#) section for more details.

When image data is received from the camera using the camera SDK, it represents raw intensities coming off the sensor. Without color processing, this will look like a monochrome mosaic or pixelated version of the scene the camera is looking at. This is because Thorlabs color cameras have a color filter in front of the sensor which will selectively allow certain wavelengths of light to pass through for each pixel. In the case of Bayer filters, each pixel is only receiving Red wavelengths, Blue wavelengths, or Green wavelengths of light. The mosaic pattern that one will see from the raw image data will correspond to this Bayer color filter pattern.

Fortunately, the camera holds all the model-specific parameters that one needs to turn this monochrome mosaic image data into a colored image. Two of those parameters are the color filter type and the color filter array phase, which specify the physical features of the color filter. The camera also contains a color correction matrix and a default white balance matrix that are used in the color processing pipeline to take the colored image and adjust the colors to more accurately reflect what is seen by humans under typical conditions. All these parameters can be used to initialize a mono to color processor for typical lighting environments. The end result is a nicely colored image that requires only a few steps to create. There are additional controls after initialization to allow the user to customize the coloring of the image for more accurate color correction.

0.1.2 Getting Started

Getting started programming with the Thorlabs mono to color processing sdk is straightforward. This guide describes how to program with the mono to color processing sdk using compiled (native) languages such as C or C++. This guide assumes the user knows how to open a camera and get an image. To learn about working with Thorlabs cameras in native code, check out the Thorlabs Camera C API Reference. This guide contains snippets of code that show short examples of how to use specific functions. The full example that these snippets are based on can be found in the [Example](#) section.

- Call `tl_mono_to_color_processing_initialize()` to dynamically load the mono to color SDK library and get handles to its exported functions.

```
/*  
 * Initialize mono to color sdk  
 */  
int error_code = tl_mono_to_color_processing_initialize();  
if (error_code) { /* Perform error handling. If exiting the program, remember to close any opened cameras or processors (and their SDKs). */ }
```

- Get all the information needed to initialize a mono to color processor:
-

- camera_sensor_type - use tl_camera_get_camera_sensor_type() to query this value from the camera.

```
// camera sensor type
enum TL_CAMERA_SENSOR_TYPE camera_sensor_type;
int error_code = tl_camera_get_camera_sensor_type(camera_handle, &camera_sensor_type);
if (error_code) { /* Perform error handling. If exiting the program, remember to close any opened cameras or processors (and their SDKs). */ }
```

- color_filter_array_phase - use tl_camera_get_color_filter_array_phase() to query this value from the camera.

```
// color filter array phase
enum TL_COLOR_FILTER_ARRAY_PHASE color_filter_array_phase;
int error_code = tl_camera_get_color_filter_array_phase(camera_handle, &color_filter_array_phase);
if (error_code) { /* Perform error handling. If exiting the program, remember to close any opened cameras or processors (and their SDKs). */ }
```

- color_correction_matrix - use tl_camera_get_color_correction_matrix() to query this value from the camera.

```
// color correction matrix
float color_correction_matrix[9];
int error_code = tl_camera_get_color_correction_matrix(camera_handle, color_correction_matrix);
if (error_code) { /* Perform error handling. If exiting the program, remember to close any opened cameras or processors (and their SDKs). */ }
```

- default_white_balance_matrix - use tl_camera_get_default_white_balance_matrix() to query this from the camera.

```
// default white balance matrix
float default_white_balance_matrix[9];
int error_code = tl_camera_get_default_white_balance_matrix(camera_handle, default_white_balance_matrix);
if (error_code) { /* Perform error handling. If exiting the program, remember to close any opened cameras or processors (and their SDKs). */ }
```

- bit_depth - use tl_camera_get_bit_depth() to query this from the camera.

```
// bit depth
int bit_depth;
int error_code = tl_camera_get_bit_depth(camera_handle, &bit_depth);
if (error_code) { /* Perform error handling. If exiting the program, remember to close any opened cameras or processors (and their SDKs). */ }
```

- Create a mono to color processor by calling tl_mono_to_color_create_mono_to_color_processor() with the above information.

```
/*
 * Create mono to color processor
 */
void *mono_to_color_processor;
int error_code = tl_mono_to_color_create_mono_to_color_processor(camera_sensor_type, color_filter_array_phase, color_correction_matrix,
default_white_balance_matrix, bit_depth, &mono_to_color_processor);
if (error_code) { /* Perform error handling. If exiting the program, remember to close any opened cameras or processors (and their SDKs). */ }
```

- After creating a mono to color processor you can adjust the output format using tl_mono_to_color_set_output_format(). You can also adjust the color space using tl_mono_to_color_set_color_space().

```
// set color space
int error_code = tl_mono_to_color_set_color_space(mono_to_color_processor, TL_MONO_TO_COLOR_SPACE_SRGB);
if (error_code) { /* Perform error handling. If exiting the program, remember to close any opened cameras or processors (and their SDKs). */ }
// set output format
int error_code = tl_mono_to_color_set_output_format(mono_to_color_processor, TL_COLOR_FORMAT_RGB_PIXEL);
if (error_code) { /* Perform error handling. If exiting the program, remember to close any opened cameras or processors (and their SDKs). */ }
```

- There are red, blue, and green gain values that can be adjusted to alter the colors of the output images. To set each of these gain values use tl_mono_to_color_set_red_gain(), tl_mono_to_color_set_blue_gain(), and tl_mono_to_color_set_green_gain() respectively. Higher gain values amplify the intensity of that color, whereas lower gain values will diminish the intensity.

```
int error_code = tl_mono_to_color_set_red_gain(mono_to_color_processor, 1.0);
if (error_code) { /* Perform error handling. If exiting the program, remember to close any opened cameras or processors (and their SDKs). */ }
int error_code = tl_mono_to_color_set_blue_gain(mono_to_color_processor, 1.0);
```

```

if (error_code) { /* Perform error handling. If exiting the program, remember to close any opened cameras or processors (and their SDKs). */ }
int error_code = tl_mono_to_color_set_green_gain(mono_to_color_processor, 1.0);
if (error_code) { /* Perform error handling. If exiting the program, remember to close any opened cameras or processors (and their SDKs). */ }

```

- Allocate memory for the colored output data. The amount of space that needs to be allocated depends on the number of pixels in the image data and the desired output bit depth. The following is an example of what size the output buffer needs to be when the image data corresponds to an image with dimensions 1920 x 1080 (2,073,600 pixels):

- For 24 bpp images, the output buffer size is: $(\text{sizeof}(\text{unsigned char}) * (2073600) * 3) = 49766400$ bytes.
- For 32 bpp images, the output buffer size is: $(\text{sizeof}(\text{unsigned char}) * (2073600) * 4) = 66355200$ bytes.
- For 48 bpp images, the output buffer size is: $(\text{sizeof}(\text{unsigned short}) * (2073600) * 3) = 99532800$ bytes;

- Call the appropriate transform function for the desired output bit depth to get the colored image:

- `tl_mono_to_color_transform_to_24()`: This will output the color image data as 8 bits per channel, 3 channels per pixel.

```

/*
 * call transform function (24bpp)
 */
// allocate output buffer
unsigned char *output_buffer_24 = malloc(sizeof(unsigned char) * image_width * image_height * 3);
// transform to 24 bpp
int error_code = tl_mono_to_color_transform_to_24(mono_to_color_processor, image_buffer, image_width, image_height, output_buffer_24);
if (error_code)
{
    /* Perform error handling. If exiting the program, remember to close any opened cameras or processors (and their SDKs), and free any additional
resources that may have been acquired.*/
}
/* Perform action using the output buffer */
// clean up
free(output_buffer_24);
output_buffer_24 = NULL;

```

- `tl_mono_to_color_transform_to_32()`: This will output the color image data as 8 bits per channel, 4 channels per pixel. The fourth channel is a byte padding that can be used as an Alpha channel.

```

/*
 * call transform function (32bpp)
 */
// allocate output buffer
unsigned char *output_buffer_32 = malloc(sizeof(unsigned char) * image_width * image_height * 4);
// transform to 32 bpp
int error_code = tl_mono_to_color_transform_to_32(mono_to_color_processor, image_buffer, image_width, image_height, output_buffer_32);
if (error_code)
{
    /* Perform error handling. If exiting the program, remember to close any opened cameras or processors (and their SDKs), and free any additional
resources that may have been acquired.*/
}
/* Perform action using the output buffer */
// clean up
free(output_buffer_32);
output_buffer_32 = NULL;

```

- `tl_mono_to_color_transform_to_48()`: This will output the color image data as 16 bits per channel, 3 channels per pixel.

```
/*
 * call transform function (48bpp)
 */
// allocate output buffer
unsigned short *output_buffer_48 = malloc(sizeof(unsigned short) * image_width * image_height * 3);
// transform to 48 bpp
int error_code = tl_mono_to_color_transform_to_48(mono_to_color_processor, image_buffer, image_width, image_height, output_buffer_48);
if (error_code)
{
    /* Perform error handling. If exiting the program, remember to close any opened cameras or processors (and their SDKs), and free any additional
resources that may have been acquired.*/
}
/* Perform action using the output buffer */
// clean up
free(output_buffer_48);
output_buffer_48 = NULL;
```

- Each transform function requires the pointers to the input buffer (the image data) and the output buffer, but they also require the width and height of the image corresponding to the data in the input buffer. It is recommended to save the image width and height to variables after arming a camera and pass those variables to the mono to color processor each time transform is called. This is because querying these values from the camera each time a transform function needs to be called will significantly slow down the image processing. It is safe to save the image width and height after arming since the camera cannot change these values while it is armed.

```
int image_width = 0;
int error_code = tl_camera_get_image_width(camera_handle, &image_width);
if (error_code) { /* Perform error handling. If exiting the program, remember to close any opened cameras or processors (and their SDKs). */ }
int image_height = 0;
int error_code = tl_camera_get_image_height(camera_handle, &image_height);
if (error_code) { /* Perform error handling. If exiting the program, remember to close any opened cameras or processors (and their SDKs). */ }
```

- When the application is finished with the mono to color processor, dispose of it using `tl_mono_to_color_destroy_mono_to_color_processor()`. Feel free to use as many mono to color processors as your application requires, just make sure they are all cleanly destroyed.

```
// destroy mono to color processor
int error_code = tl_mono_to_color_destroy_mono_to_color_processor(mono_to_color_processor);
if (error_code) { /* Perform error handling. If exiting the program, remember to close any opened cameras or processors (and their SDKs). */ }
```

- When the application is finished with the mono to color processing sdk, dispose of it using `tl_mono_to_color_processing_module_terminate()`.

```
// close mono to color sdk
int error_code = tl_mono_to_color_processing_terminate();
if (error_code) { /* Perform error handling. If exiting the program, remember to close any opened cameras or processors (and their SDKs). */ }
```

0.1.3 Example

The following code shows how to use a mono to color processor to get a color image. The code is meant to be a working example that goes through the entire imaging process, starting from opening a Thorlabs camera.

```
/* Mono to Color Example
 *
 * This example opens a camera and polls for an image. The image is transformed into 48, 32, and 24 bits per pixel
```



```
* color images.
*
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "tl_camera_sdk.h"
#include "tl_camera_sdk_load.h"
#include "tl_color_enum.h"
#include "tl_mono_to_color_enum.h"
#include "tl_mono_to_color_processing_load.h"
void *camera_handle;
// Close the camera, then the camera SDK, then the camera dll.
int close_camera_sdk_and_dll(void) {
    int error_code = 0;
    int return_code = 0;
    if(camera_handle)
    {
        error_code = tl_camera_close_camera(camera_handle);
        if(error_code)
        {
            printf("Failed to close camera!\n%s\n", tl_camera_get_last_error());
            return_code = error_code;
        }
    }
    error_code = tl_camera_close_sdk();
    if (error_code)
    {
        printf("Failed to close SDK!\n");
        return_code = error_code;
    }
    error_code = tl_camera_sdk_dll_terminate();
    if (error_code)
    {
        printf("Failed to close dll!\n");
        return_code = error_code;
    }
    if (!return_code) printf("SDK & dll closed\n");
    return return_code;
}
int main(void)
{
    int error_code = 0;
    /*
    * Camera initialization
    */
    // Initializes dll
    error_code = tl_camera_sdk_dll_initialize();
    if (error_code)
    {
        printf("Failed to initialize dll!\n");
        return 1;
    }
    else printf("Successfully initialized dll\n");
    // Open the SDK
    error_code = tl_camera_open_sdk();
```

```

if (error_code)
{
    printf("Failed to open SDK!\n");
    tl_camera_sdk_dll_terminate();
    return 1;
}
else printf("Successfully opened SDK\n");
char camera_ids[1024];
// Discover cameras.
error_code = tl_camera_discover_available_cameras(camera_ids, 1024);
if (error_code)
{
    printf("Failed to get available cameras!\n%s\n", tl_camera_get_last_error());
    close_camera_sdk_and_dll();
    return 1;
}
else printf("camera IDs: %s\n", camera_ids);
// Check for no cameras.
if (!strlen(camera_ids))
{
    printf("Error: did not find any cameras!\n");
    close_camera_sdk_and_dll();
    return 1;
}
// Camera IDs are separated by spaces.
char* p_space = strchr(camera_ids, ' ');
if (p_space)
{
    *p_space = '\0'; // isolate the first detected camera
}
char first_camera[256];
// Copy the ID of the first camera to separate buffer (for clarity)
strcpy(first_camera, camera_ids);
printf("First camera_id = %s\n", first_camera);
// Connect to the camera (get a handle to it).
if (tl_camera_open_camera(first_camera, &camera_handle))
{
    printf("Failed to open camera!\n%s\n", tl_camera_get_last_error());
    close_camera_sdk_and_dll();
    return 1;
}
printf("Camera handle = 0x%p\n", camera_handle);
// Arm the camera
error_code = tl_camera_arm(camera_handle, 2);
if (error_code)
{
    printf("Failed to arm camera!\n%s\n", tl_camera_get_last_error());
    close_camera_sdk_and_dll();
    return 1;
}
/*
 * Get image width and height
 */
int image_width = 0;
int image_height = 0;
error_code = tl_camera_get_image_width(camera_handle, &image_width);

```

```
if (error_code)
{
    printf("Failed to get image width!\n%s\n", tl_camera_get_last_error());
    close_camera_sdk_and_dll();
    return 1;
}
error_code = tl_camera_get_image_height(camera_handle, &image_height);
if (error_code)
{
    printf("Failed to get image height!\n%s\n", tl_camera_get_last_error());
    close_camera_sdk_and_dll();
    return 1;
}
/*
 * Initialize all parameters needed to create a mono to color processor
 */
// color filter array phase
enum TL_COLOR_FILTER_ARRAY_PHASE color_filter_array_phase;
error_code = tl_camera_get_color_filter_array_phase(camera_handle, &color_filter_array_phase);
if (error_code)
{
    printf("Failed to get color filter array phase from the camera!\n%s\n", tl_camera_get_last_error());
    close_camera_sdk_and_dll();
    return 1;
}
// camera sensor type
enum TL_CAMERA_SENSOR_TYPE camera_sensor_type;
error_code = tl_camera_get_camera_sensor_type(camera_handle, &camera_sensor_type);
if (error_code)
{
    printf("Failed to get sensor type from the camera!\n%s\n", tl_camera_get_last_error());
    close_camera_sdk_and_dll();
    return 1;
}
// color correction matrix
float color_correction_matrix[9];
error_code = tl_camera_get_color_correction_matrix(camera_handle, color_correction_matrix);
if (error_code)
{
    printf("Failed to get color correction matrix from the camera!\n%s\n", tl_camera_get_last_error());
    close_camera_sdk_and_dll();
    return 1;
}
// default white balance matrix
float default_white_balance_matrix[9];
error_code = tl_camera_get_default_white_balance_matrix(camera_handle, default_white_balance_matrix);
if (error_code)
{
    printf("Failed to get default white balance matrix from the camera!\n%s\n", tl_camera_get_last_error());
    close_camera_sdk_and_dll();
    return 1;
}
// bit depth
int bit_depth;
error_code = tl_camera_get_bit_depth(camera_handle, &bit_depth);
if (error_code)
```

```
{
    printf("Failed to get bit depth from the camera!\n%s\n", tl_camera_get_last_error());
    close_camera_sdk_and_dll();
    return 1;
}
/*
 * Initialize mono to color sdk
 */
error_code = tl_mono_to_color_processing_initialize();
if(error_code)
{
    printf("Failed to initialize mono to color processing module!\n");
    close_camera_sdk_and_dll();
    return 1;
}
/*
 * Create mono to color processor
 */
void *mono_to_color_processor;
error_code = tl_mono_to_color_create_mono_to_color_processor(camera_sensor_type, color_filter_array_phase, color_correction_matrix, default_white_balance_matrix,
    bit_depth, &mono_to_color_processor);
if (error_code)
{
    printf("Failed to create mono to color processor!\n%s\n", tl_mono_to_color_get_last_error());
    error_code = tl_mono_to_color_processing_terminate();
    if(error_code)
    {
        printf("Failed to terminate mono to color processing module!\n");
    }
    close_camera_sdk_and_dll();
    return 1;
}
/*
 * Setup any mono to color processor properties
 */
// set color space
error_code = tl_mono_to_color_set_color_space(mono_to_color_processor, TL_MONO_TO_COLOR_SPACE_SRGB);
if (error_code)
{
    printf("Failed to set color space!\n%s\n", tl_mono_to_color_get_last_error());
    error_code = tl_mono_to_color_processing_terminate();
    if (error_code)
    {
        printf("Failed to terminate mono to color processing module!\n");
    }
    close_camera_sdk_and_dll();
    return 1;
}
// set output format
error_code = tl_mono_to_color_set_output_format(mono_to_color_processor, TL_COLOR_FORMAT_RGB_PIXEL);
if (error_code)
{
    printf("Failed to set output format!\n%s\n", tl_mono_to_color_get_last_error());
    error_code = tl_mono_to_color_processing_terminate();
    if (error_code)
    {
```

```
        printf("Failed to terminate mono to color processing module!\n");
    }
    close_camera_sdk_and_dll();
    return 1;
}
/*
 * Acquire an image from the camera
 */
// initialize frame variables
unsigned short *image_buffer = 0;
int frame_count = 0;
unsigned char *metadata = 0;
int metadata_size_in_bytes = 0;
// Send software trigger
error_code = tl_camera_issue_software_trigger(camera_handle);
if(error_code)
{
    printf("Failed to issue software trigger!\n%s\n", tl_camera_get_last_error());
    error_code = tl_mono_to_color_processing_terminate();
    if (error_code)
    {
        printf("Failed to terminate mono to color processing module!\n");
    }
    close_camera_sdk_and_dll();
    return 1;
}
// Poll for an image
while (!image_buffer)
{
    error_code = tl_camera_get_pending_frame_or_null(camera_handle, &image_buffer, &frame_count, &metadata, &metadata_size_in_bytes);
    if (error_code) // error codes are nonzero values
    {
        printf("Error while trying to get pending frame!\n%s\n", tl_camera_get_last_error());
        break;
    }
}
if(!image_buffer)
{
    printf("Failed to get an image!\n");
    error_code = tl_mono_to_color_destroy_mono_to_color_processor(mono_to_color_processor);
    if (error_code)
    {
        printf("Failed to destroy mono to color processor!\n%s\n", tl_mono_to_color_get_last_error());
    }
    error_code = tl_mono_to_color_processing_terminate();
    if (error_code)
    {
        printf("Failed to terminate mono to color processing module!\n");
    }
    close_camera_sdk_and_dll();
    return 1;
}
// disarm camera
error_code = tl_camera_disarm(camera_handle);
if(error_code)
{

```

```
printf("Failed to disarm camera!\n%s\n", tl_camera_get_last_error());
error_code = tl_mono_to_color_processing_terminate();
if (error_code)
{
    printf("Failed to terminate mono to color processing module!\n");
}
close_camera_sdk_and_dll();
return 1;
}
/*
 * call transform function (48bpp)
 */
// allocate output buffer
unsigned short *output_buffer_48 = malloc(sizeof(unsigned short) * image_width * image_height * 3);
// transform to 48 bpp
error_code = tl_mono_to_color_transform_to_48(mono_to_color_processor, image_buffer, image_width, image_height, output_buffer_48);
if (error_code)
{
    printf("Failed transform monochrome image to 48bpp!\n%s\n", tl_mono_to_color_get_last_error());
    error_code = tl_mono_to_color_destroy_mono_to_color_processor(mono_to_color_processor);
    if (error_code)
    {
        printf("Failed to destroy mono to color processor!\n%s\n", tl_mono_to_color_get_last_error());
    }
    error_code = tl_mono_to_color_processing_terminate();
    if (error_code)
    {
        printf("Failed to terminate mono to color processing module!\n");
    }
    close_camera_sdk_and_dll();
    free(output_buffer_48);
    return 1;
}
/* Perform action using the output buffer */
free(output_buffer_48);
/*
 * call transform function (32bpp)
 */
// allocate output buffer
unsigned char *output_buffer_32 = malloc(sizeof(unsigned char) * image_width * image_height * 4);
// transform to 32 bpp
error_code = tl_mono_to_color_transform_to_32(mono_to_color_processor, image_buffer, image_width, image_height, output_buffer_32);
if (error_code)
{
    printf("Failed transform monochrome image to 32bpp!\n%s\n", tl_mono_to_color_get_last_error());
    error_code = tl_mono_to_color_destroy_mono_to_color_processor(mono_to_color_processor);
    if (error_code)
    {
        printf("Failed to destroy mono to color processor!\n%s\n", tl_mono_to_color_get_last_error());
    }
    error_code = tl_mono_to_color_processing_terminate();
    if (error_code)
    {
        printf("Failed to terminate mono to color processing module!\n");
    }
}
close_camera_sdk_and_dll();
```

```
    free(output_buffer_32);
    return 1;
}
/* Perform action using the output buffer */
free(output_buffer_32);
/*
 * call transform function (24bpp)
 */
// allocate output buffer
unsigned char *output_buffer_24 = malloc(sizeof(unsigned char) * image_width * image_height * 3);
// transform to 24 bpp
error_code = tl_mono_to_color_transform_to_24(mono_to_color_processor, image_buffer, image_width, image_height, output_buffer_24);
if (error_code)
{
    printf("Failed transform monochrome image to 24bpp!\n%s\n", tl_mono_to_color_get_last_error());
    error_code = tl_mono_to_color_destroy_mono_to_color_processor(mono_to_color_processor);
    if (error_code)
    {
        printf("Failed to destroy mono to color processor!\n%s\n", tl_mono_to_color_get_last_error());
    }
    error_code = tl_mono_to_color_processing_terminate();
    if (error_code)
    {
        printf("Failed to terminate mono to color processing module!\n");
    }
    close_camera_sdk_and_dll();
    free(output_buffer_24);
    return 1;
}
/* Perform action using the output buffer */
free(output_buffer_24);
/*
 * Clean up
 */
// destroy mono to color processor
error_code = tl_mono_to_color_destroy_mono_to_color_processor(mono_to_color_processor);
if (error_code)
{
    printf(" Failed to destroy mono to color processor!\n%s\n", tl_mono_to_color_get_last_error());
}
//close mono to color sdk
error_code = tl_mono_to_color_processing_terminate();
if (error_code)
{
    printf(" Failed to terminate mono to color processing module!\n");
}
// close the camera
close_camera_sdk_and_dll();
}
```

0.1.4 Advanced Color Processing

The mono to color processing sdk utilizes the color processing suite to transform monochrome images. The color processing suite is made up of a couple SDKs and a number of DLLs that perform specific functions in the color processing pipeline. A color processing pipeline itself is made up of a number of steps, each with customizable parameters. The color processing suite allows for total customization of this processing pipeline. Instead of exposing all these controls, the mono to color processing sdk wraps up the color processing suite into a single sdk with a reduced API. It sets the pipeline up for a very common use case, but allows the details to still be controlled using high level properties like color space and color gains. This will be useful for a large majority of users who are looking to get a colored image.

For advanced users looking to dive into the details of color processing pipelines, the color processing suite is available for use. The mono to color processing sdk can be used alongside the color processing suite, but to reduce confusion it is recommended to use one or the other. See the Thorlabs Scientific Color Processing Suite documentation to learn about what is possible with the color processing suite.

0.2 File Index

0.2.1 File List

Here is a list of all documented files with brief descriptions:

tl_mono_to_color_enum.h	This file includes the declarations of all the enumerations unique to the mono to color processing module	12
tl_mono_to_color_processing.h	This file includes the declaration prototypes of all the API functions contained in the mono to color processing module	15

0.3 File Documentation

0.3.1 tl_mono_to_color_enum.h File Reference

Enumerations

- enum [TL_MONO_TO_COLOR_SPACE](#) { [TL_MONO_TO_COLOR_SPACE_SRGB](#), [TL_MONO_TO_COLOR_SPACE_LINEAR_SRGB](#), [TL_MONO_TO_COLOR_SPACE_MAX](#) }
- enum [TL_MONO_TO_COLOR_ERROR](#) { [TL_MONO_TO_COLOR_ERROR_NONE](#), [TL_MONO_TO_COLOR_ERROR_COLOR_PROCESSING_ERROR](#), [TL_MONO_TO_COLOR_ERROR_DEMOSAIC_ERROR](#), [TL_MONO_TO_COLOR_ERROR_CAMERA_ERROR](#), [TL_MONO_TO_COLOR_ERROR_INITIALIZATION_ERROR](#), [TL_MONO_TO_COLOR_ERROR_NULL_INSTANCE](#), [TL_MONO_TO_COLOR_ERROR_UNKNOWN_ERROR](#), [TL_MONO_TO_COLOR_ERROR_INVALID_INPUT](#), [TL_MONO_TO_COLOR_ERROR_RUNTIME_ERROR](#), [TL_MONO_TO_COLOR_ERROR_TERMINATION_ERROR](#), [TL_MONO_TO_COLOR_ERROR_MAX](#) }

0.3.1.1 Enumeration Type Documentation

0.3.1.1.1 TL_MONO_TO_COLOR_ERROR

enum [TL_MONO_TO_COLOR_ERROR](#)

The TL_MONO_TO_COLOR_ERROR enumeration lists all possible error codes that can be returned from the mono to color processing library.

Enumerator

TL_MONO_TO_COLOR_ERROR_NONE	The command succeeded with no errors.
TL_MONO_TO_COLOR_ERROR_COLOR_PROCESSING_ERROR	An error was thrown in the color processing module.
TL_MONO_TO_COLOR_ERROR_DEMOSAIC_ERROR	An error was thrown in the demosaic module.
TL_MONO_TO_COLOR_ERROR_CAMERA_ERROR	An error was thrown in the camera module.
TL_MONO_TO_COLOR_ERROR_INITIALIZATION_ERROR	An error was thrown during module initialization.
TL_MONO_TO_COLOR_ERROR_NULL_INSTANCE	A pointer that was about to be dereferenced was null.
TL_MONO_TO_COLOR_ERROR_UNKNOWN_ERROR	An unknown error occurred.
TL_MONO_TO_COLOR_ERROR_INVALID_INPUT	An unacceptable input was passed to this function.
TL_MONO_TO_COLOR_ERROR_RUNTIME_ERROR	Mono to color processor caught an invalid action.
TL_MONO_TO_COLOR_ERROR_TERMINATION_ERROR	An error was thrown during module termination.
TL_MONO_TO_COLOR_ERROR_MAX	A sentinel value (DO NOT USE).

0.3.1.1.2 TL_MONO_TO_COLOR_SPACE

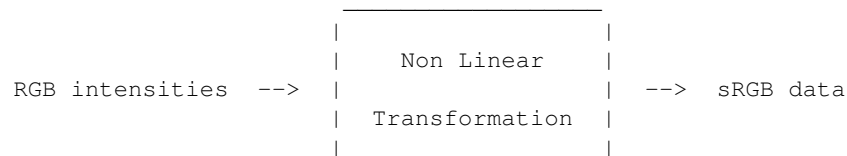
enum [TL_MONO_TO_COLOR_SPACE](#)

The TL_MONO_TO_COLOR_SPACE enumeration lists all the supported color spaces in the mono to color processing module.

A color space describes how the colors in an image are going to be specified. Some commonly used color spaces are those derived from the RGB color model, in which each pixel has a Red, Blue, and Green component. This means the amount of color that can be expressed in a single pixel is all the possible combinations of Red, Blue, and Green. If we assume the image data is in bytes, each component can take any value from 0 to 255. The total number of colors that a pixel could express can be calculated as $256 * 256 * 256 = 16777216$ different colors.

There are many different color spaces that are used for different purposes. The mono to color processor supports two color spaces that are both derived from the RGB color model: sRGB and Linear sRGB.

sRGB or standard Red Green Blue is a common color space used for displaying images on computer monitors or for sending images over the internet. In addition to the Red, Blue, and Green components combining to define the color of a pixel, the final RGB values undergo a nonlinear transformation to be put in the sRGB color space. The exact transfer function can be found online by searching for the sRGB specification. The purpose of this transformation is to represent the colors in a way that looks more accurate to humans.



Linear sRGB is very similar to sRGB, but does not perform the non linear transformation. The transformation of the data in sRGB changes the RGB intensities, whereas this color space is much more representative of the raw image data coming off the sensor. Without the transformation however, images in the Linear sRGB color space do not look as accurate as those in sRGB. When deciding between Linear sRGB and sRGB, use Linear sRGB when the actual intensities of the raw image data are important, and use sRGB when the image needs to look accurate to the human eye.

Enumerator

TL_MONO_TO_COLOR_SPACE_SRGB	The sRGB color space.
TL_MONO_TO_COLOR_SPACE_LINEAR_SRGB	The Linear sRGB color space.
TL_MONO_TO_COLOR_SPACE_MAX	A sentinel value (DO NOT USE).

0.3.2 tl_mono_to_color_processing.h File Reference

```
#include "tl_mono_to_color_enum.h"
#include "tl_color_enum.h"
#include "tl_camera_sdk.h"
```

Typedefs

- typedef int(* [TL_MONO_TO_COLOR_PROCESSING_MODULE_INITIALIZE](#)) (void)
 - typedef int(* [TL_MONO_TO_COLOR_CREATE_MONO_TO_COLOR_PROCESSOR](#)) (enum TL_CAMERA_SENSOR_TYPE, enum TL_COLOR_FILTER_ARRAY_PHASE, float *, float *, int, void **)
 - typedef int(* [TL_MONO_TO_COLOR_DESTROY_MONO_TO_COLOR_PROCESSOR](#)) (void *)
 - typedef int(* [TL_MONO_TO_COLOR_GET_COLOR_SPACE](#)) (void *, enum [TL_MONO_TO_COLOR_SPACE](#) *)
 - typedef int(* [TL_MONO_TO_COLOR_SET_COLOR_SPACE](#)) (void *, enum [TL_MONO_TO_COLOR_SPACE](#))
 - typedef int(* [TL_MONO_TO_COLOR_GET_OUTPUT_FORMAT](#)) (void *, enum TL_COLOR_FORMAT *)
 - typedef int(* [TL_MONO_TO_COLOR_SET_OUTPUT_FORMAT](#)) (void *, enum TL_COLOR_FORMAT)
 - typedef int(* [TL_MONO_TO_COLOR_GET_RED_GAIN](#)) (void *, float *)
 - typedef int(* [TL_MONO_TO_COLOR_SET_RED_GAIN](#)) (void *, float)
 - typedef int(* [TL_MONO_TO_COLOR_GET_BLUE_GAIN](#)) (void *, float *)
 - typedef int(* [TL_MONO_TO_COLOR_SET_BLUE_GAIN](#)) (void *, float)
 - typedef int(* [TL_MONO_TO_COLOR_GET_GREEN_GAIN](#)) (void *, float *)
 - typedef int(* [TL_MONO_TO_COLOR_SET_GREEN_GAIN](#)) (void *, float)
 - typedef int(* [TL_MONO_TO_COLOR_TRANSFORM_TO_48](#)) (void *, unsigned short *, int, int, unsigned short *)
 - typedef int(* [TL_MONO_TO_COLOR_TRANSFORM_TO_32](#)) (void *, unsigned short *, int, int, unsigned char *)
 - typedef int(* [TL_MONO_TO_COLOR_TRANSFORM_TO_24](#)) (void *, unsigned short *, int, int, unsigned char *)
 - typedef int(* [TL_MONO_TO_COLOR_PROCESSING_MODULE_TERMINATE](#)) (void)
 - typedef const char *(* [TL_MONO_TO_COLOR_GET_LAST_ERROR](#)) (void)
 - typedef int(* [TL_MONO_TO_COLOR_GET_CAMERA_SENSOR_TYPE](#)) (void *, enum TL_CAMERA_SENSOR_TYPE *)
 - typedef int(* [TL_MONO_TO_COLOR_GET_COLOR_FILTER_ARRAY_PHASE](#)) (void *, enum TL_COLOR_FILTER_ARRAY_PHASE *)
 - typedef int(* [TL_MONO_TO_COLOR_GET_COLOR_CORRECTION_MATRIX](#)) (void *, float *)
 - typedef int(* [TL_MONO_TO_COLOR_GET_DEFAULT_WHITE_BALANCE_MATRIX](#)) (void *, float *)
 - typedef int(* [TL_MONO_TO_COLOR_GET_BIT_DEPTH](#)) (void *, int *)
-

0.3.2.1 Typedef Documentation

0.3.2.1.1 TL_MONO_TO_COLOR_CREATE_MONO_TO_COLOR_PROCESSOR

```
typedef int (* TL_MONO_TO_COLOR_CREATE_MONO_TO_COLOR_PROCESSOR) (enum TL_CAMERA_SENSOR_TYPE, enum TL_COLOR_FILTER_ARRAY_PHASE, float *, float *,  
int, void **)
```

This function creates a mono to color processor instance and returns a pointer to that instance.

The mono to color processing instance is a handle to the internal mono to color processing state.

Parameters

<i>camera_sensor_type</i>	The camera sensor type that will be used. This value should be queried from the camera.
<i>color_filter_array_phase</i>	The color filter array phase that will be used. This value should be queried from the camera.
<i>color_correction_matrix</i>	The color correction matrix that will be used. This value should be queried from the camera.
<i>default_white_balance_matrix</i>	The default white balance matrix that will be used. The initial values of the color gains are derived from this matrix. This value should be queried from the camera.
<i>bit_depth</i>	The desired bit depth for this mono to color processor. This value should be queried from the camera.
<i>mono_to_color_handle</i>	Pointer to a pointer to the new mono to color processor instance.

Returns

[TL_MONO_TO_COLOR_ERROR](#) value to indicate success or failure ([TL_MONO_TO_COLOR_ERROR_NONE](#) indicates success).

0.3.2.1.2 TL_MONO_TO_COLOR_DESTROY_MONO_TO_COLOR_PROCESSOR

```
typedef int (* TL_MONO_TO_COLOR_DESTROY_MONO_TO_COLOR_PROCESSOR) (void *)
```

This function destroys a mono to color processor instance.

After this function is called, the mono to color processor cannot be used. It is advised to set the mono_to_color_processor pointer to NULL to avoid accidental usage after this function has been called. The pointer does not need to be deleted. This function must be called for each mono to color processor before the program exits to cleanly release any open resources.

Parameters

<i>mono_to_color_handle</i>	A pointer to a mono to color processor handle.
-----------------------------	--

Returns

[TL_MONO_TO_COLOR_ERROR](#) value to indicate success or failure ([TL_MONO_TO_COLOR_ERROR_NONE](#) indicates success).

0.3.2.1.3 TL_MONO_TO_COLOR_GET_BIT_DEPTH

```
typedef int (* TL_MONO_TO_COLOR_GET_BIT_DEPTH) (void *, int *)
```

This function allows the user to retrieve the bit depth associated with the given color processor. This value is set during construction and cannot be modified during the lifetime of a mono to color object.

Parameters

<i>mono_to_color_handle</i>	A pointer to a mono to color processor handle.
<i>bit_depth</i>	The bit depth is returned to the caller by reference.

Returns

A [TL_MONO_TO_COLOR_ERROR](#) value to indicate success or failure ([TL_MONO_TO_COLOR_ERROR_NONE](#) indicates success).

0.3.2.1.4 TL_MONO_TO_COLOR_GET_BLUE_GAIN

```
typedef int(* TL_MONO_TO_COLOR_GET_BLUE_GAIN) (void *, float *)
```

This function allows the caller to get the blue gain of the mono to color processor.

The blue gain is used to amplify or diminish the blue component of images transformed by the mono to color processor. Lower values correspond to a less intense blue component and higher values correspond to a more intense blue component. The red, blue, and green gain values can be used to color-correct images, such as white balancing an image. The default value for blue gain is determined by the default white balance matrix that is given to the mono to color processor during construction.

Parameters

<i>mono_to_color_handle</i>	A pointer to a mono to color processor handle.
<i>blue_gain</i>	A pointer to a float value. The blue gain is returned by reference.

Returns

A [TL_MONO_TO_COLOR_ERROR](#) value to indicate success or failure ([TL_MONO_TO_COLOR_ERROR_NONE](#) indicates success).

0.3.2.1.5 TL_MONO_TO_COLOR_GET_CAMERA_SENSOR_TYPE

```
typedef int(* TL_MONO_TO_COLOR_GET_CAMERA_SENSOR_TYPE) (void *, enum TL_CAMERA_SENSOR_TYPE *)
```

This function allows the user to retrieve the camera sensor type that has been associated with the given mono to color processor. This value is set during construction and cannot be modified during the lifetime of a mono to color object.

Parameters

<i>mono_to_color_handle</i>	A pointer to a mono to color processor handle.
<i>camera_sensor_type</i>	The camera sensor type returned to the caller by reference.

Returns

A [TL_MONO_TO_COLOR_ERROR](#) value to indicate success or failure ([TL_MONO_TO_COLOR_ERROR_NONE](#) indicates success).

0.3.2.1.6 TL_MONO_TO_COLOR_GET_COLOR_CORRECTION_MATRIX

```
typedef int(* TL_MONO_TO_COLOR_GET_COLOR_CORRECTION_MATRIX) (void *, float *)
```

This function allows the user to retrieve the color correction matrix that has been associated with the given mono to color processor. The color correction matrix is a 3x3 matrix that is represented by a float array of 9 elements. This value is set during construction and cannot be modified during the lifetime of a mono to color object.

Parameters

<i>mono_to_color_handle</i>	A pointer to a mono to color processor handle.
<i>color_correction_matrix</i>	The color correction matrix from the mono to color processor will be copied into this array. This array must have a size of at least 9.

Returns

A [TL_MONO_TO_COLOR_ERROR](#) value to indicate success or failure ([TL_MONO_TO_COLOR_ERROR_NONE](#) indicates success).

0.3.2.1.7 TL_MONO_TO_COLOR_GET_COLOR_FILTER_ARRAY_PHASE

```
typedef int(* TL_MONO_TO_COLOR_GET_COLOR_FILTER_ARRAY_PHASE) (void *, enum TL_COLOR_FILTER_ARRAY_PHASE *)
```

This function allows the user to retrieve the color filter array phase associated with the given mono to color processor. This value is set during construction and cannot be modified during the lifetime of a mono to color object.

Parameters

<i>mono_to_color_handle</i>	A pointer to a mono to color processor handle.
<i>color_filter_array_phase</i>	A pointer to a ::TL_COLOR_FILTER_ARRAY_PHASE. The color filter array phase is returned to the caller by reference.

Returns

A [TL_MONO_TO_COLOR_ERROR](#) value to indicate success or failure ([TL_MONO_TO_COLOR_ERROR_NONE](#) indicates success).

0.3.2.1.8 TL_MONO_TO_COLOR_GET_COLOR_SPACE

```
typedef int(* TL_MONO_TO_COLOR_GET_COLOR_SPACE) (void *, enum TL\_MONO\_TO\_COLOR\_SPACE *)
```

This function allows the caller to get the color space of the mono to color processor.

The color space property specifies how the colors will be represented when output from the mono to color processor. The default value is [TL_MONO_TO_COLOR_SPACE_SRGB](#).

Parameters

<i>mono_to_color_handle</i>	A pointer to a mono to color processor handle.
<i>color_space</i>	A pointer to a TL_MONO_TO_COLOR_SPACE . The color space is returned by reference.

Returns

[TL_MONO_TO_COLOR_ERROR](#) value to indicate success or failure ([TL_MONO_TO_COLOR_ERROR_NONE](#) indicates success).

0.3.2.1.9 TL_MONO_TO_COLOR_GET_DEFAULT_WHITE_BALANCE_MATRIX

```
typedef int(* TL_MONO_TO_COLOR_GET_DEFAULT_WHITE_BALANCE_MATRIX) (void *, float *)
```

This function allows the user to retrieve the default white balance matrix that has been associated with the given mono to color processor. The default white balance matrix is a 3x3 matrix that is represented by a float array of 9 elements. This value is set during construction and cannot be modified during the lifetime of a mono to color object.

Parameters

<i>mono_to_color_handle</i>	A pointer to a mono to color processor handle.
<i>default_white_balance_matrix</i>	The default white balance matrix from the mono to color processor will be copied into this array. This array must have a size of at least 9.

Returns

A [TL_MONO_TO_COLOR_ERROR](#) value to indicate success or failure ([TL_MONO_TO_COLOR_ERROR_NONE](#) indicates success).

0.3.2.1.10 TL_MONO_TO_COLOR_GET_GREEN_GAIN

```
typedef int(* TL_MONO_TO_COLOR_GET_GREEN_GAIN) (void *, float *)
```

This function allows the caller to get the green gain of the mono to color processor.

The green gain is used to amplify or diminish the green component of images transformed by the mono to color processor. Lower values correspond to a less intense green component and higher values correspond to a more intense green component. The red, blue, and green gain values can be used to color-correct images, such as white balancing an image. The default value for green gain is determined by the default white balance matrix that is given to the mono to color processor during construction.

Parameters

<i>mono_to_color_handle</i>	A pointer to a mono to color processor handle.
<i>green_gain</i>	A pointer to a float value. The green gain is returned by reference.

Returns

A [TL_MONO_TO_COLOR_ERROR](#) value to indicate success or failure ([TL_MONO_TO_COLOR_ERROR_NONE](#) indicates success).

0.3.2.1.11 TL_MONO_TO_COLOR_GET_LAST_ERROR

```
typedef const char*(* TL_MONO_TO_COLOR_GET_LAST_ERROR) (void)
```

This function returns a human-friendly string of that corresponds to the last error that occurred.

Returns

A pointer to a Null-terminated const char string that contains the error message.

0.3.2.1.12 TL_MONO_TO_COLOR_GET_OUTPUT_FORMAT

```
typedef int(* TL_MONO_TO_COLOR_GET_OUTPUT_FORMAT) (void *, enum TL_COLOR_FORMAT *)
```

This function allows the caller to get the output format of the mono to color processor.

The output format property specifies the ordering of the data that is output from the mono to color processor. For example, RGB_PIXEL will layout pixel data in sets of 3 corresponding to Red, Green, and Blue components: RGBRGBRGBRGB..., Whereas BGR_PLANAR will layout all the Blue components, then all the Green components, and then all the Red components: BBBB... GGGG... RRRR... .

The default value is ::TL_COLOR_FORMAT_RGB_PIXEL.

Parameters

<i>mono_to_color_handle</i>	A pointer to a mono to color processor handle.
<i>color_space</i>	A pointer to a ::TL_COLOR_FORMAT. The output format is returned by reference.

Returns

A [TL_MONO_TO_COLOR_ERROR](#) value to indicate success or failure ([TL_MONO_TO_COLOR_ERROR_NONE](#) indicates success).

0.3.2.1.13 TL_MONO_TO_COLOR_GET_RED_GAIN

```
typedef int(* TL_MONO_TO_COLOR_GET_RED_GAIN) (void *, float *)
```

This function allows the caller to get the red gain of the mono to color processor.

The red gain is used to amplify or diminish the red component of images transformed by the mono to color processor. Lower values correspond to a less intense red component and higher values correspond to a more intense red component. The red, blue, and green gain values can be used to color-correct images, such as white balancing an image. The default value for red gain is determined by the default white balance matrix that is given to the mono to color processor during construction.

Parameters

<i>mono_to_color_handle</i>	A pointer to a mono to color processor handle.
<i>red_gain</i>	A pointer to a float value. The red gain is returned by reference.

Returns

[TL_MONO_TO_COLOR_ERROR](#) value to indicate success or failure ([TL_MONO_TO_COLOR_ERROR_NONE](#) indicates success).

0.3.2.1.14 TL_MONO_TO_COLOR_PROCESSING_MODULE_INITIALIZE

```
typedef int (* TL_MONO_TO_COLOR_PROCESSING_MODULE_INITIALIZE) (void)
```

This function initializes the mono to color processing module. It must be called prior to calling any other mono to color processing module API function.

Returns

[TL_MONO_TO_COLOR_ERROR](#) value to indicate success or failure ([TL_MONO_TO_COLOR_ERROR_NONE](#) indicates success).

0.3.2.1.15 TL_MONO_TO_COLOR_PROCESSING_MODULE_TERMINATE

```
typedef int (* TL_MONO_TO_COLOR_PROCESSING_MODULE_TERMINATE) (void)
```

This function terminates the mono to color processing module. After this function is called, `tl_mono_to_color_` functions will no longer be defined. This function must be called before the program is finished to cleanly dispose of any open resources.

Returns

A [TL_MONO_TO_COLOR_ERROR](#) value to indicate success or failure ([TL_MONO_TO_COLOR_ERROR_NONE](#) indicates success).

0.3.2.1.16 TL_MONO_TO_COLOR_SET_BLUE_GAIN

```
typedef int (* TL_MONO_TO_COLOR_SET_BLUE_GAIN) (void *, float)
```

This function allows the caller to set the blue gain of the mono to color processor.

Parameters

<i>mono_to_color_handle</i>	A pointer to a mono to color processor handle.
<i>blue_gain</i>	A float value that the mono to color processor will use to set its blue gain property.

Returns

A [TL_MONO_TO_COLOR_ERROR](#) value to indicate success or failure ([TL_MONO_TO_COLOR_ERROR_NONE](#) indicates success).

0.3.2.1.17 TL_MONO_TO_COLOR_SET_COLOR_SPACE

```
typedef int (* TL_MONO_TO_COLOR_SET_COLOR_SPACE) (void *, enum TL\_MONO\_TO\_COLOR\_SPACE)
```

This function allows the caller to set the color space of the mono to color processor.

Parameters

<i>mono_to_color_handle</i>	A pointer to a mono to color processor handle.
<i>color_space</i>	A TL_MONO_TO_COLOR_SPACE that the mono to color will use to set its color space property.

Returns

A [TL_MONO_TO_COLOR_ERROR](#) value to indicate success or failure ([TL_MONO_TO_COLOR_ERROR_NONE](#) indicates success).

0.3.2.1.18 TL_MONO_TO_COLOR_SET_GREEN_GAIN

```
typedef int(* TL_MONO_TO_COLOR_SET_GREEN_GAIN) (void *, float)
```

This function allows the caller to set the green gain of the mono to color processor.

Parameters

<i>mono_to_color_handle</i>	A pointer to a mono to color processor handle.
<i>green_gain</i>	A float value that the mono to color processor will use to set its green gain property.

Returns

A [TL_MONO_TO_COLOR_ERROR](#) value to indicate success or failure ([TL_MONO_TO_COLOR_ERROR_NONE](#) indicates success).

0.3.2.1.19 TL_MONO_TO_COLOR_SET_OUTPUT_FORMAT

```
typedef int(* TL_MONO_TO_COLOR_SET_OUTPUT_FORMAT) (void *, enum TL_COLOR_FORMAT)
```

This function allows the caller to set the output format of the mono to color processor.

Parameters

<i>mono_to_color_handle</i>	A pointer to a mono to color processor handle.
<i>output_format</i>	A ::TL_COLOR_FORMAT that the mono to color processor will use to set its color space property.

Returns

A [TL_MONO_TO_COLOR_ERROR](#) value to indicate success or failure ([TL_MONO_TO_COLOR_ERROR_NONE](#) indicates success).

0.3.2.1.20 TL_MONO_TO_COLOR_SET_RED_GAIN

```
typedef int(* TL_MONO_TO_COLOR_SET_RED_GAIN) (void *, float)
```

This function allows the caller to set the red gain of the mono to color processor.

Parameters

<i>mono_to_color_handle</i>	A pointer to a mono to color processor handle.
<i>red_gain</i>	A float value that the mono to color processor will use to set its red gain property.

Returns

[TL_MONO_TO_COLOR_ERROR](#) value to indicate success or failure ([TL_MONO_TO_COLOR_ERROR_NONE](#) indicates success).

0.3.2.1.21 TL_MONO_TO_COLOR_TRANSFORM_TO_24

```
typedef int(* TL_MONO_TO_COLOR_TRANSFORM_TO_24) (void *, unsigned short *, int, int, unsigned char *)
```

This function transforms a monochrome image into a color image with 24 bits per pixel, where each pixel contains 3 channels and each channel is 8 bits. The number of elements in the output buffer should be 3x that of the input buffer, but the data type should be unsigned char instead of unsigned short.

Parameters

<i>mono_to_color_handle</i>	A pointer to a mono to color processor handle.
<i>input_buffer</i>	The monochrome image data from the camera.
<i>image_width</i>	The width in pixels associated with the image data in <i>input_buffer</i> .
<i>image_height</i>	The height in pixels associated with the image data in <i>input_buffer</i> .
<i>output_buffer</i>	The output buffer containing the colored image created by transforming the image data in <i>input_buffer</i> .

Returns

A [TL_MONO_TO_COLOR_ERROR](#) value to indicate success or failure ([TL_MONO_TO_COLOR_ERROR_NONE](#) indicates success).

0.3.2.1.22 TL_MONO_TO_COLOR_TRANSFORM_TO_32

```
typedef int(* TL_MONO_TO_COLOR_TRANSFORM_TO_32) (void *, unsigned short *, int, int, unsigned char *)
```

This function transforms a monochrome image into a color image with 32 bits per pixel, where each pixel contains 4 channels and each channel is 8 bits. The 4th channel will be an Alpha channel with all values set to 0 (0% opacity). For example, if the output format is RGB_PIXEL, then the resulting structure of the output data will be RGBARGBARGBARGBA... .

The number of elements in the output buffer should be 4x that of the input buffer, but the data type should be unsigned char instead of unsigned short.

Parameters

<i>mono_to_color_handle</i>	A pointer to a mono to color processor handle.
<i>input_buffer</i>	The monochrome image data from the camera.
<i>image_width</i>	The width in pixels associated with the image data in <i>input_buffer</i> .
<i>image_height</i>	The height in pixels associated with the image data in <i>input_buffer</i> .
<i>output_buffer</i>	The output buffer containing the colored image created by transforming the image data in <i>input_buffer</i> .

Returns

A [TL_MONO_TO_COLOR_ERROR](#) value to indicate success or failure ([TL_MONO_TO_COLOR_ERROR_NONE](#) indicates success).

0.3.2.1.23 TL_MONO_TO_COLOR_TRANSFORM_TO_48

```
typedef int(* TL_MONO_TO_COLOR_TRANSFORM_TO_48) (void *, unsigned short *, int, int, unsigned short *)
```

This function transforms a monochrome image into a color image with 48 bits per pixel, where each pixel contains 3 channels and each channel is 16 bits. The number of elements in the output buffer should be 3x that of the input buffer.

Parameters

<i>mono_to_color_handle</i>	A pointer to a mono to color processor handle.
<i>input_buffer</i>	The monochrome image data from the camera.
<i>image_width</i>	The width in pixels associated with the image data in input_buffer.
<i>image_height</i>	The height in pixels associated with the image data in input_buffer.
<i>output_buffer</i>	The output buffer containing the colored image created by transforming the image data in input_buffer.

Returns

A [TL_MONO_TO_COLOR_ERROR](#) value to indicate success or failure ([TL_MONO_TO_COLOR_ERROR_NONE](#) indicates success).

Index

TL_MONO_TO_COLOR_CREATE_MONO_TO_COLOR_PROCESSOR
 tl_mono_to_color_processing.h, [16](#)
TL_MONO_TO_COLOR_DESTROY_MONO_TO_COLOR_PROCESSOR
 tl_mono_to_color_processing.h, [16](#)
tl_mono_to_color_enum.h, [12](#)
 TL_MONO_TO_COLOR_ERROR, [13](#)
 TL_MONO_TO_COLOR_ERROR_CAMERA_ERROR, [13](#)
 TL_MONO_TO_COLOR_ERROR_COLOR_PROCESSING_ERROR, [13](#)
 TL_MONO_TO_COLOR_ERROR_DEMOSAIC_ERROR, [13](#)
 TL_MONO_TO_COLOR_ERROR_INITIALIZATION_ERROR, [13](#)
 TL_MONO_TO_COLOR_ERROR_INVALID_INPUT, [13](#)
 TL_MONO_TO_COLOR_ERROR_MAX, [13](#)
 TL_MONO_TO_COLOR_ERROR_NONE, [13](#)
 TL_MONO_TO_COLOR_ERROR_NULL_INSTANCE, [13](#)
 TL_MONO_TO_COLOR_ERROR_RUNTIME_ERROR, [13](#)
 TL_MONO_TO_COLOR_ERROR_TERMINATION_ERROR, [13](#)
 TL_MONO_TO_COLOR_ERROR_UNKNOWN_ERROR, [13](#)
 TL_MONO_TO_COLOR_SPACE, [13](#)
 TL_MONO_TO_COLOR_SPACE_LINEAR_SRGB, [14](#)
 TL_MONO_TO_COLOR_SPACE_MAX, [14](#)
 TL_MONO_TO_COLOR_SPACE_SRGB, [14](#)
TL_MONO_TO_COLOR_ERROR
 tl_mono_to_color_enum.h, [13](#)
TL_MONO_TO_COLOR_ERROR_CAMERA_ERROR
 tl_mono_to_color_enum.h, [13](#)
TL_MONO_TO_COLOR_ERROR_COLOR_PROCESSING_ERROR
 tl_mono_to_color_enum.h, [13](#)
TL_MONO_TO_COLOR_ERROR_DEMOSAIC_ERROR
 tl_mono_to_color_enum.h, [13](#)

TL_MONO_TO_COLOR_ERROR_INITIALIZATION_ERROR
 tl_mono_to_color_enum.h, [13](#)
TL_MONO_TO_COLOR_ERROR_INVALID_INPUT
 tl_mono_to_color_enum.h, [13](#)
TL_MONO_TO_COLOR_ERROR_MAX
 tl_mono_to_color_enum.h, [13](#)
TL_MONO_TO_COLOR_ERROR_NONE
 tl_mono_to_color_enum.h, [13](#)
TL_MONO_TO_COLOR_ERROR_NULL_INSTANCE
 tl_mono_to_color_enum.h, [13](#)
TL_MONO_TO_COLOR_ERROR_RUNTIME_ERROR
 tl_mono_to_color_enum.h, [13](#)
TL_MONO_TO_COLOR_ERROR_TERMINATION_ERROR
 tl_mono_to_color_enum.h, [13](#)
TL_MONO_TO_COLOR_ERROR_UNKNOWN_ERROR
 tl_mono_to_color_enum.h, [13](#)
TL_MONO_TO_COLOR_GET_BIT_DEPTH
 tl_mono_to_color_processing.h, [17](#)
TL_MONO_TO_COLOR_GET_BLUE_GAIN
 tl_mono_to_color_processing.h, [18](#)
TL_MONO_TO_COLOR_GET_CAMERA_SENSOR_TYPE
 tl_mono_to_color_processing.h, [18](#)
TL_MONO_TO_COLOR_GET_COLOR_CORRECTION_MATRIX
 tl_mono_to_color_processing.h, [19](#)
TL_MONO_TO_COLOR_GET_COLOR_FILTER_ARRAY_PHASE
 tl_mono_to_color_processing.h, [19](#)
TL_MONO_TO_COLOR_GET_COLOR_SPACE
 tl_mono_to_color_processing.h, [20](#)
TL_MONO_TO_COLOR_GET_DEFAULT_WHITE_BALANCE_MATRIX

tl_mono_to_color_processing.h, 21
TL_MONO_TO_COLOR_GET_GREEN_GAIN
tl_mono_to_color_processing.h, 21
TL_MONO_TO_COLOR_GET_LAST_ERROR
tl_mono_to_color_processing.h, 22
TL_MONO_TO_COLOR_GET_OUTPUT_FORMAT
tl_mono_to_color_processing.h, 22
TL_MONO_TO_COLOR_GET_RED_GAIN
tl_mono_to_color_processing.h, 23
tl_mono_to_color_processing.h, 15
TL_MONO_TO_COLOR_CREATE_MONO_TO_COLOR_PROCESSOR, 16
TL_MONO_TO_COLOR_DESTROY_MONO_TO_COLOR_PROCESSOR, 16
TL_MONO_TO_COLOR_GET_BIT_DEPTH, 17
TL_MONO_TO_COLOR_GET_BLUE_GAIN, 18
TL_MONO_TO_COLOR_GET_CAMERA_SENSOR_TYPE, 18
TL_MONO_TO_COLOR_GET_COLOR_CORRECTION_MATRIX, 19
TL_MONO_TO_COLOR_GET_COLOR_FILTER_ARRAY_PHASE, 19
TL_MONO_TO_COLOR_GET_COLOR_SPACE, 20
TL_MONO_TO_COLOR_GET_DEFAULT_WHITE_BALANCE_MATRIX, 21
TL_MONO_TO_COLOR_GET_GREEN_GAIN, 21
TL_MONO_TO_COLOR_GET_LAST_ERROR, 22
TL_MONO_TO_COLOR_GET_OUTPUT_FORMAT, 22
TL_MONO_TO_COLOR_GET_RED_GAIN, 23
TL_MONO_TO_COLOR_PROCESSING_MODULE_INITIALIZE, 23
TL_MONO_TO_COLOR_PROCESSING_MODULE_TERMINATE, 24
TL_MONO_TO_COLOR_SET_BLUE_GAIN, 24
TL_MONO_TO_COLOR_SET_COLOR_SPACE, 25
TL_MONO_TO_COLOR_SET_GREEN_GAIN, 25
TL_MONO_TO_COLOR_SET_OUTPUT_FORMAT, 26
TL_MONO_TO_COLOR_SET_RED_GAIN, 27
TL_MONO_TO_COLOR_TRANSFORM_TO_24, 27
TL_MONO_TO_COLOR_TRANSFORM_TO_32, 28
TL_MONO_TO_COLOR_TRANSFORM_TO_48, 29
TL_MONO_TO_COLOR_PROCESSING_MODULE_INITIALIZE
tl_mono_to_color_processing.h, 23
TL_MONO_TO_COLOR_PROCESSING_MODULE_TERMINATE
tl_mono_to_color_processing.h, 24
TL_MONO_TO_COLOR_SET_BLUE_GAIN
tl_mono_to_color_processing.h, 24
TL_MONO_TO_COLOR_SET_COLOR_SPACE
tl_mono_to_color_processing.h, 25
TL_MONO_TO_COLOR_SET_GREEN_GAIN
tl_mono_to_color_processing.h, 25
TL_MONO_TO_COLOR_SET_OUTPUT_FORMAT
tl_mono_to_color_processing.h, 26
TL_MONO_TO_COLOR_SET_RED_GAIN
tl_mono_to_color_processing.h, 27
TL_MONO_TO_COLOR_SPACE
tl_mono_to_color_enum.h, 13
TL_MONO_TO_COLOR_SPACE_LINEAR_SRGB
tl_mono_to_color_enum.h, 14
TL_MONO_TO_COLOR_SPACE_MAX
tl_mono_to_color_enum.h, 14
TL_MONO_TO_COLOR_SPACE_SRGB
tl_mono_to_color_enum.h, 14
TL_MONO_TO_COLOR_TRANSFORM_TO_24
tl_mono_to_color_processing.h, 27
TL_MONO_TO_COLOR_TRANSFORM_TO_32
tl_mono_to_color_processing.h, 28
TL_MONO_TO_COLOR_TRANSFORM_TO_48
tl_mono_to_color_processing.h, 29
