Thorlabs Camera C API Reference

Rev. 2.0.2 2022-02-25 ITN003543-D01

Contents

0.1 Thorlabs Camera C API Reference

0.1.1 Getting Started

Getting started programming Thorlabs scientific cameras is straightforward. This guide describes how to program Thorlabs scientific cameras using compiled languages such as C or C++. Separate guides are available for Python, C#/VB.Net, LabVIEW, or MATLAB languages.

NOTE: This guide shows function type definitions in ALL_CAPITAL_LETTERS and the corresponding function pointers in all_lowercase_letters. For example, after calling tl_camera_sdk_dll_initialize, the function pointer tl_camera_open_sdk is available with a type definition of TL_CAMERA_OPEN_SDK.

All camera and mono-to-color functions are thread-safe, so no additional thread-locking is required.

There are two approaches to getting images from a camera: Poll for images from any thread or register a callback that is automatically invoked on a worker thread whenever images are received from the camera. The following steps describe the order in which the APIs should be called. It is important to perform all clean-up steps at the end to avoid crashes.

- Call tl_camera_sdk_dll_initialize to dynamically load the SDK DLL and get handles to its exported functions. Ensure that all required DLLs are in the same folder as the executable or discoverable in the Windows PATH variable.
- Initialize the SDK by calling tl_camera_open_sdk.
- Discover the serial numbers of all connected cameras by calling tl_camera_discover_available_cameras. This step must be called at least once before attempting to open any camera.
- Open a connection to a camera by calling tl_camera_open_camera with its serial number.
- Set camera properties like exposure, operation mode, and number of frames per trigger by calling the appropriate API functions such as tl_camera_set_exposure, tl_↩ camera_set_operation_mode, and tl_camera_set_frames_per_trigger_zero_for_unlimited, respectively.
- Optionally, register a callback function to enable your application to receive a notification when an image is available by calling tl_camera_set_frame_available_callback. IMPORTANT: This notification will arrive on a worker thread, and blocking this thread too long will force incoming frames to drop.
- Prepare the camera to send images to your application by calling tl_camera_arm. After arming, only a few parameters can be set such as exposure time. See the individual parameters for which ones are available once armed.
- Command the camera to start delivering images by calling tl_camera_issue_software_trigger or by setting up the camera to respond to external hardware triggers by setting the operation mode.

- If a callback is registered, then the SDK will invoke the registered callback each time an image is received; otherwise, poll for an image in a loop or on a timer using tl_camera_get_pending_frame_or_null.
- When your application has completed acquiring images, stop the camera by calling tl_camera_disarm.
- Close the connection to the camera by calling tl_camera_close_camera.
- Clean up and release SDK resources by calling tl_camera_close_sdk.
- Call tl_camera_sdk_dll_terminate to unload the SDK DLL.

0.2 File Index

0.2.1 File List

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

0.3 File Documentation

0.3.1 tl_camera_sdk.h File Reference

```
#include <stddef.h>
#include "tl_color_enum.h"
#include "tl_polarization_processor_enums.h"
```

Typedefs

- typedef int(* TL_CAMERA_ARM) (void *tl_camera_handle, int number_of_frames_to_buffer)
- typedef int(* TL_CAMERA_CLOSE_CAMERA) (void *tl_camera_handle)
- typedef int(* TL_CAMERA_CLOSE_SDK) (void)
- typedef void(* TL_CAMERA_CONNECT_CALLBACK) (char *cameraSerialNumber, enum TL_CAMERA_USB_PORT_TYPE usb_port_type, void *context)
- typedef int(* TL_CAMERA_CONVERT_DECIBELS_TO_GAIN) (void *tl_camera_handle, double gain_dB, int *index_of_gain_value)
- typedef int(* TL_CAMERA_CONVERT_GAIN_TO_DECIBELS) (void *tl_camera_handle, int index_of_gain_value, double *gain_dB)
- typedef int(* TL_CAMERA_DISARM) (void *tl_camera_handle)
- typedef void(* TL_CAMERA_DISCONNECT_CALLBACK) (char *cameraSerialNumber, void *context)
- typedef int(* TL_CAMERA_DISCOVER_AVAILABLE_CAMERAS) (char *serial_numbers, int str_length)
- typedef void(* TL_CAMERA_FRAME_AVAILABLE_CALLBACK) (void *tl_camera_handle_sender, unsigned short *image_buffer, int frame_count, unsigned char *metadata, int metadata_size_in_bytes, void *context)
- typedef int(* TL_CAMERA_GET_BINX) (void *tl_camera_handle, int *binx)
- typedef int(* TL_CAMERA_GET_BINX_RANGE) (void *tl_camera_handle, int *hbin_min, int *hbin_max)
- typedef int(* TL_CAMERA_GET_BINY) (void *tl_camera_handle, int *biny)
- typedef int(* TL_CAMERA_GET_BINY_RANGE) (void *tl_camera_handle, int *vbin_min, int *vbin_max)
- typedef int(* TL_CAMERA_GET_BIT_DEPTH) (void *tl_camera_handle, int *pixel_bit_depth)
- typedef int(* TL_CAMERA_GET_BLACK_LEVEL) (void *tl_camera_handle, int *black_level)
- typedef int(* TL_CAMERA_GET_BLACK_LEVEL_RANGE) (void *tl_camera_handle, int *min, int *max)
- typedef int(* TL_CAMERA_GET_CAMERA_COLOR_CORRECTION_MATRIX_OUTPUT_COLOR_SPACE) (void *tl_camera_handle, char *output_color_space)
- typedef int(* TL_CAMERA_GET_CAMERA_SENSOR_TYPE) (void *tl_camera_handle, enum TL_CAMERA_SENSOR_TYPE *camera_sensor_type)
- typedef int(* TL_CAMERA_GET_COLOR_CORRECTION_MATRIX) (void *tl_camera_handle, float *matrix)
- typedef int(* TL_CAMERA_GET_COLOR_FILTER_ARRAY_PHASE) (void *tl_camera_handle, enum TL_COLOR_FILTER_ARRAY_PHASE *cfaPhase)
- typedef int(* TL_CAMERA_GET_IS_COOLING_ENABLED) (void *tl_camera_handle, int *is_cooling_enabled)
- typedef int(* TL_CAMERA_GET_DATA_RATE) (void *tl_camera_handle, enum TL_CAMERA_DATA_RATE *data_rate)
- typedef int(* TL_CAMERA_GET_DEFAULT_WHITE_BALANCE_MATRIX) (void *tl_camera_handle, float *matrix)
- typedef int(* TL_CAMERA_GET_EXPOSURE_TIME) (void *tl_camera_handle, long long *exposure_time_us)
- typedef int(* TL_CAMERA_GET_EXPOSURE_TIME_RANGE) (void *tl_camera_handle, long long *exposure_time_us_min, long long *exposure_time_us_max)
- typedef int(* TL_CAMERA_GET_FIRMWARE_VERSION) (void *tl_camera_handle, char *firmware_version, int str_length)
- typedef int(* TL_CAMERA_GET_FRAME_AVAILABLE_CALLBACK) (void *tl_camera_handle, TL_CAMERA_FRAME_AVAILABLE_CALLBACK *handler)
- typedef int(* TL_CAMERA_GET_FRAME_RATE_CONTROL_VALUE) (void *tl_camera_handle, double *frame_rate_fps)
- typedef int(* TL_CAMERA_GET_FRAME_TIME) (void *tl_camera_handle, int *frame_time_us)
- typedef int(* TL_CAMERA_GET_IMAGE_POLL_TIMEOUT) (void *tl_camera_handle, int *timeout_ms)
- typedef int(* TL_CAMERA_GET_COMMUNICATION_INTERFACE) (void *tl_camera_handle, enum TL_CAMERA_COMMUNICATION_INTERFACE *communication_↩ interface)

- typedef int(* TL_CAMERA_GET_PENDING_FRAME_OR_NULL) (void *tl_camera_handle, unsigned short **image_buffer, int *frame_count, unsigned char **metadata, int *metadata_size_in_bytes)
- typedef int(* TL_CAMERA_GET_POLAR_PHASE) (void *tl_camera_handle, enum TL_POLARIZATION_PROCESSOR_POLAR_PHASE *polar_phase)
- typedef int(* TL_CAMERA_GET_ROI) (void *tl_camera_handle, int *upper_left_x_pixels, int *upper_left_y_pixels, int *lower_right_x_pixels, int *lower_right_y_pixels)
- typedef int(* TL_CAMERA_GET_ROI_RANGE) (void *tl_camera_handle, int *upper_left_x_pixels_min, int *upper_left_y_pixels_min, int *lower_right_x_pixels_min, int *lower_right_y_pixels_min, int *upper_left_x_pixels_max, int *upper_left_y_pixels_max, int *lower_right_x_pixels_max, int *lower_right_y_pixels_max)
- typedef int(* TL_CAMERA_GET_SENSOR_HEIGHT) (void *tl_camera_handle, int *height_pixels)
- typedef int(* TL_CAMERA_GET_SENSOR_PIXEL_HEIGHT) (void *tl_camera_handle, double *pixel_height_um)
- typedef int(* TL_CAMERA_GET_SENSOR_PIXEL_SIZE_BYTES) (void *tl_camera_handle, int *sensor_pixel_size_bytes)
- typedef int(* TL_CAMERA_GET_SENSOR_PIXEL_WIDTH) (void *tl_camera_handle, double *pixel_width_um)
- typedef int(* TL_CAMERA_GET_SENSOR_READOUT_TIME) (void *tl_camera_handle, int *sensor_readout_time_ns)
- typedef int(* TL_CAMERA_GET_SENSOR_WIDTH) (void *tl_camera_handle, int *width_pixels)
- typedef int(* TL_CAMERA_GET_SERIAL_NUMBER) (void *tl_camera_handle, char *serial_number, int str_length)
- typedef int(* TL_CAMERA_GET_SERIAL_NUMBER_STRING_LENGTH_RANGE) (void *tl_camera_handle, int *serial_number_min, int *serial_number_max)
- typedef int(* TL_CAMERA_GET_TAP_BALANCE_ENABLE) (void *tl_camera_handle, int *taps_balance_enable)
- typedef int(* TL_CAMERA_GET_TAPS) (void *tl_camera_handle, enum TL_CAMERA_TAPS *taps)
- typedef int(* TL_CAMERA_GET_TIMESTAMP_CLOCK_FREQUENCY) (void *tl_camera_handle, int *timestamp_clock_frequency_hz_or_zero)
- typedef int(* TL_CAMERA_GET_TRIGGER_POLARITY) (void *tl_camera_handle, enum TL_CAMERA_TRIGGER_POLARITY *trigger_polarity_enum)
- typedef int(* TL_CAMERA_GET_USB_PORT_TYPE) (void *tl_camera_handle, enum TL_CAMERA_USB_PORT_TYPE *usb_port_type)
- typedef int(* TL_CAMERA_GET_USER_MEMORY) (void *tl_camera_handle, unsigned char *destination_data_buffer, long long number_of_bytes_to_read, long long camera_user_memory_offset_bytes)
- typedef int(* TL_CAMERA_GET_USER_MEMORY_MAXIMUM_SIZE) (void *tl_camera_handle, long long *maximum_size_bytes)
- typedef int(* TL_CAMERA_ISSUE_SOFTWARE_TRIGGER) (void *tl_camera_handle)
- typedef int(* TL_CAMERA_OPEN_CAMERA) (char *camera_serial_number, void **tl_camera_handle)
- typedef int(* TL_CAMERA_OPEN_SDK) (void)
- typedef int(* TL_CAMERA_SET_BINX) (void *tl_camera_handle, int binx)
- typedef int(* TL_CAMERA_SET_BINY) (void *tl_camera_handle, int biny)
- typedef int(* TL_CAMERA_SET_BLACK_LEVEL) (void *tl_camera_handle, int black_level)
- typedef int(* TL_CAMERA_SET_CAMERA_CONNECT_CALLBACK) (TL_CAMERA_CONNECT_CALLBACK handler, void *context)
- typedef int(* TL_CAMERA_SET_CAMERA_DISCONNECT_CALLBACK) (TL_CAMERA_DISCONNECT_CALLBACK handler, void *context)
- typedef int(* TL_CAMERA_SET_DATA_RATE) (void *tl_camera_handle, enum TL_CAMERA_DATA_RATE data_rate)
- typedef int(* TL_CAMERA_SET_EXPOSURE_TIME) (void *tl_camera_handle, long long exposure_time_us)
- typedef int(* TL_CAMERA_SET_FRAME_AVAILABLE_CALLBACK) (void *tl_camera_handle, TL_CAMERA_FRAME_AVAILABLE_CALLBACK handler, void *context)
- typedef int(* TL_CAMERA_SET_FRAME_RATE_CONTROL_VALUE) (void *tl_camera_handle, double frame_rate_fps)
- typedef int(* TL_CAMERA_SET_FRAMES_PER_TRIGGER_ZERO_FOR_UNLIMITED) (void *tl_camera_handle, unsigned int number_of_frames_per_trigger_or_zero↩_for_unlimited)

Enumerations

- enum TL_CAMERA_OPERATION_MODE { TL_CAMERA_OPERATION_MODE_SOFTWARE_TRIGGERED, TL_CAMERA_OPERATION_MODE_HARDWARE_TRIGGERED, TL_CAMERA_OPERATION_MODE_BULB, TL_CAMERA_OPERATION_MODE_RESERVED1, TL_CAMERA_OPERATION_MODE_RESERVED2, TL_CAMERA_OPERATION_MODE_MAX }
- enum TL_CAMERA_SENSOR_TYPE { TL_CAMERA_SENSOR_TYPE_MONOCHROME, TL_CAMERA_SENSOR_TYPE_BAYER, TL_CAMERA_SENSOR_TYPE_MONOCHROME_POLA TL_CAMERA_SENSOR_TYPE_MAX }
- enum TL_CAMERA_TAPS { TL_CAMERA_TAPS_SINGLE_TAP = 0, TL_CAMERA_TAPS_DUAL_TAP = 1, TL_CAMERA_TAPS_QUAD_TAP = 2, TL_CAMERA_TAPS_MAX_TAP = 3 }
- enum TL_CAMERA_TRIGGER_POLARITY { TL_CAMERA_TRIGGER_POLARITY_ACTIVE_HIGH, TL_CAMERA_TRIGGER_POLARITY_ACTIVE_LOW, TL_CAMERA_TRIGGER_POLAR }
- enum TL_CAMERA_USB_PORT_TYPE { TL_CAMERA_USB_PORT_TYPE_USB1_0, TL_CAMERA_USB_PORT_TYPE_USB2_0, TL_CAMERA_USB_PORT_TYPE_USB3_0, TL_CAMERA_USB_PORT_TYPE_MAX }

0.3.1.1 Typedef Documentation

0.3.1.1.1 TL_CAMERA_ARM

```
typedef int(* TL_CAMERA_ARM) (void *tl_camera_handle, int number_of_frames_to_buffer)
```

Before issuing software or hardware triggers to get images from a camera, prepare it for imaging by calling tl_camera_arm.

Depending on the desired trigger type, either call tl_camera_issue_software_trigger or issue a hardware trigger.

To start a camera in continuous mode:

1. Ensure that the camera is not armed.
2. Set the operation mode to software triggered.

3. Set the number of frames per trigger to 0 (which indicates continuous operation from a single trigger).

4. Arm the camera.

5. Issue a single software trigger. The camera will then self-trigger frames until tl_camera_disarm() is called.

   To start a camera for hardware triggering:

1. Ensure that the camera is not armed.

2. Set the operation mode to hardware or bulb triggered.

3. Set the number of frames per trigger to 1.

4. Set the trigger polarity to rising- or falling-edge triggering.

5. Arm the camera.

6. Issue a hardware-trigger signal on the trigger input.

Parameters

tl_camera_handle	The handle to the camera to arm.
number_of_frames_to_buffer	The number of frames to allocate in the internal image buffer. For most use cases, this should be set to 2, which allows one image to be transferring from the camera to the buffer while the other is being read out.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.2 TL_CAMERA_CLOSE_CAMERA

```
typedef int(* TL_CAMERA_CLOSE_CAMERA) (void *tl_camera_handle)
```

Terminates the host application's connection to the specified camera.

This function releases platform resources used by the camera's software abstraction and generally cleans up state associated with the specified camera. After calling this function, the specified handle is invalid and must NOT be used for any further camera interaction.

Any attempt to do so is not permitted and could result in undefined behavior.

Parameters

tl_camera_handle	The handle to the camera to close.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.3 TL_CAMERA_CLOSE_SDK

```
typedef int(* TL_CAMERA_CLOSE_SDK) (void)
```

Releases any platform resources that were used by the SDK and generally cleans up SDK state.

This function must be called by the user application prior to exiting.

Any attempt to call an API function after this function has been called is not permitted and could result in undefined behavior.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.4 TL_CAMERA_CONNECT_CALLBACK

```
typedef void(* TL_CAMERA_CONNECT_CALLBACK) (char *cameraSerialNumber, enum TL_CAMERA_USB_PORT_TYPE usb_port_type, void *context)
```

Enables the user application to register for notifications of device connect events.

The SDK will invoke the registered callback function every time a device is connected to the computer.

Parameters

cameraSerialNumber	The serial number of the connected device.
usb_port_type	The USB port type that the device was connected to.
context	A pointer to a user specified context. This parameter is ignored by the SDK.

0.3.1.1.5 TL_CAMERA_CONVERT_DECIBELS_TO_GAIN

```
typedef int(* TL_CAMERA_CONVERT_DECIBELS_TO_GAIN) (void *tl_camera_handle, double gain_dB, int *index_of_gain_value)
```

Converts the decibel value received from tl_camera_convert_gain_to_decibels back into a gain index value. This function will return the closest index to the specified decibel gain.

Parameters

tl_camera_handle	The camera handle.
gain_dB	The decibel value to convert.
index_of_gain_value	A reference to receive the gain index value.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.6 TL_CAMERA_CONVERT_GAIN_TO_DECIBELS

```
typedef int(* TL_CAMERA_CONVERT_GAIN_TO_DECIBELS) (void *tl_camera_handle, int index_of_gain_value, double *gain_dB)
```

The gain value is set in the camera with tl_camera_set_gain. It is retrieved from the camera with tl_camera_get_gain. The range of possible gain values varies by camera model. It can be retrieved from the camera with tl_camera_get_gain_range. The gain value units vary by camera model, but with this conversion function, it can be converted to decibels (dB).

Parameters

tl_camera_handle	The camera handle.
index_of_gain_value	The gain value returned by tl_camera_get_gain.
gain_dB	A reference to receive the gain value in decibels (dB).

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.7 TL_CAMERA_DISARM

```
typedef int(* TL_CAMERA_DISARM) (void *tl_camera_handle)
```

When finished issuing software or hardware triggers, call tl_camera_disarm(). This will cause a camera in continuous image delivery mode to stop delivering images and will reset the camera's internal image delivery state.

This allows setting parameters are are not available in armed mode such as ROI and binning.

Parameters

tl_camera_handle	The camera handle associated with the disarm request.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.8 TL_CAMERA_DISCONNECT_CALLBACK

```
typedef void(* TL_CAMERA_DISCONNECT_CALLBACK) (char *cameraSerialNumber, void *context)
```

Enables the user application to register for notifications of device disconnect events.

The SDK will invoke the registered callback function every time a device is disconnected from the computer.

Parameters

cameraSerialNumber	The serial number of the disconnected device.
context	A pointer to a user specified context. This parameter is ignored by the SDK.

0.3.1.1.9 TL_CAMERA_DISCOVER_AVAILABLE_CAMERAS

```
typedef int(* TL_CAMERA_DISCOVER_AVAILABLE_CAMERAS) (char *serial_numbers, int str_length)
```

Returns a space character delimited list of detected camera serial numbers.

This step is required before opening a camera even if the serial number is already known.

Parameters

serial_numbers	A pointer to a character string to receive the serial number list.
str_length	The size in bytes of the character buffer specified in the serial_numbers parameter.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.10 TL_CAMERA_FRAME_AVAILABLE_CALLBACK

```
typedef void(* TL_CAMERA_FRAME_AVAILABLE_CALLBACK) (void *tl_camera_handle_sender, unsigned short *image_buffer, int frame_count, unsigned char
*metadata, int metadata_size_in_bytes, void *context)
```

Register a callback function for notifications of frame (image) availability.

NOTE: There are two methods for getting image frames from the camera: Polling or registering for a callback.

1. Poll with the tl_camera_get_pending_frame_or_null function, typically from the main thread (polling from any thread is valid).

2. Register for a callback. In this case, frames will arrive on a worker thread to avoid interrupting the main thread. Be sure to use proper thread-locking techniques if the data needs to be marshaled from the worker thread to the main thread (such as for display in a graphical user interface).

The SDK will invoke the registered callback function every time an image is received from the camera.

IMPORTANT: The memory is allocate inside the camera SDK. The provided pointer is only valid for the duration of this callback. Either complete copy the data before returning from this callback or complete all tasks related to the image before returning.

The data for both color and monochrome cameras are ordered left to right across a row followed by the next row below it.

For monochrome cameras, each pixel requires two bytes.

For color cameras, each pixel requires two bytes for blue followed by two bytes for green followed by two bytes for red (BBGGRRBBGGRRBBGGRR...). Therefore, each color pixel requires six bytes.

Parameters

tl_camera_handle_sender	The instance of the tl_camera sending the event.
image_buffer	The pointer to the buffer that contains the image data. The byte ordering of the data in this buffer is little-endian. IMPORTANT: This pointer is only valid for the duration of this callback.
frame_count	The image count corresponding to the received image during the current acquisition run. If the image metadata section was not found, this will be 0.
metadata	The pointer to the buffer that contains the image metadata. The byte ordering of the data in this buffer is little-endian. If the metadata section was not found, this will be null. IMPORTANT: This pointer is only valid for the duration of this callback. For details about the metadata, please see tl_camera_get_pending_frame_or_null.
metadata_size_in_bytes	The size (in bytes) of the image metadata buffer. If the metadata section was not found, this will be 0.
context	A pointer to a user specified context. This parameter is ignored by the SDK.

For more information on the image metadata format, click here: IMAGE_METADATA_DOCUMENTATION

0.3.1.1.11 TL_CAMERA_GET_BINX

```
typedef int(* TL_CAMERA_GET_BINX) (void *tl_camera_handle, int *binx)
```

Binning sums adjacent sensor pixels into "super pixels". It trades off spatial resolution for sensitivity and speed. For example, if a sensor is 1920 by 1080 pixels and binning is set to two in the X direction and two in the Y direction, the resulting image will be 960 by 540 pixels. Since smaller images require less data to be transmitted to the host computer,

binning may increase the frame rate. By default, binning is set to one in both horizontal and vertical directions.

Gets the current horizontal binning value for the specified camera.

Parameters

tl_camera_handle	The camera handle associated with the horizontal binning request.
binx	A reference to receive the current horizontal binning value.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.12 TL_CAMERA_GET_BINX_RANGE

```
typedef int(* TL_CAMERA_GET_BINX_RANGE) (void *tl_camera_handle, int *hbin_min, int *hbin_max)
```

Binning sums adjacent sensor pixels into "super pixels". It trades off spatial resolution for sensitivity and speed. For example, if a sensor is 1920 by 1080 pixels and binning is set to two in the X direction and two in the Y direction, the resulting image will be 960 by 540 pixels. Since smaller images require less data to be transmitted to the host computer, binning may increase the frame rate. By default, binning is set to one in both horizontal and vertical directions.

Gets the range of acceptable values for the horizontal (adjacent pixels in the X direction) binning setting for the specified camera.

Parameters

tl_camera_handle	The camera handle associated with the horizontal binning range request.
hbin_min	A reference to receive the minimum acceptable value for horizontal binning.
hbin_max	A reference to receive the maximum acceptable value for horizontal binning.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.13 TL_CAMERA_GET_BINY

```
typedef int(* TL_CAMERA_GET_BINY) (void *tl_camera_handle, int *biny)
```

Binning sums adjacent sensor pixels into "super pixels". It trades off spatial resolution for sensitivity and speed. For example, if a sensor is 1920 by 1080 pixels and binning is set to two in the X direction and two in the Y direction, the resulting image will be 960 by 540 pixels. Since smaller images require less data to be transmitted to the host computer, binning may increase the frame rate. By default, binning is set to one in both horizontal and vertical directions.

Gets the current vertical binning value for the specified camera.

Parameters

tl_camera_handle	The camera handle associated with the vertical binning request.
biny	A reference to receive the current vertical binning value.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.14 TL_CAMERA_GET_BINY_RANGE

```
typedef int(* TL_CAMERA_GET_BINY_RANGE) (void *tl_camera_handle, int *vbin_min, int *vbin_max)
```

Binning sums adjacent sensor pixels into "super pixels". It trades off spatial resolution for sensitivity and speed. For example, if a sensor is 1920 by 1080 pixels and binning is set to two in the X direction and two in the Y direction, the resulting image will be 960 by 540 pixels. Since smaller images require less data to be transmitted to the host computer, binning may increase the frame rate. By default, binning is set to one in both horizontal and vertical directions.

Gets the range of acceptable values for the vertical (adjacent pixels in the Y direction) binning setting for the specified camera.

Parameters

tl_camera_handle	The camera handle associated with the vertical binning range request.
vbin_min	A reference to receive the minimum acceptable value for vertical binning.
vbin_max	A reference to receive the maximum acceptable value for vertical binning.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.15 TL_CAMERA_GET_BIT_DEPTH

```
typedef int(* TL_CAMERA_GET_BIT_DEPTH) (void *tl_camera_handle, int *pixel_bit_depth)
```

The number of bits to which a pixel value is digitized on a camera.

In the image data that is delivered to the host application, the bit depth indicates how many of the lower bits of each 16-bit value are relevant.

Parameters

tl_camera_handle	The camera handle associated with the pixel bit depth request.
pixel_bit_depth	A reference to receive the current pixel bit depth.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.16 TL_CAMERA_GET_BLACK_LEVEL

```
typedef int(* TL_CAMERA_GET_BLACK_LEVEL) (void *tl_camera_handle, int *black_level)
```

Gets the current black level value for the specified camera.

Parameters

tl_camera_handle	The camera handle associated with the black level request.
black_level	A reference to receive the current black level value.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.17 TL_CAMERA_GET_BLACK_LEVEL_RANGE

```
typedef int(* TL_CAMERA_GET_BLACK_LEVEL_RANGE) (void *tl_camera_handle, int *min, int *max)
```

Gets the black level maximum value for the specified camera.

Parameters

tl_camera_handle	The camera handle associated with the black level command.
min	A reference to receive the black level minimum value.
max	A reference to receive the black level maximum value.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.18 TL_CAMERA_GET_CAMERA_COLOR_CORRECTION_MATRIX_OUTPUT_COLOR_SPACE

```
typedef int(* TL_CAMERA_GET_CAMERA_COLOR_CORRECTION_MATRIX_OUTPUT_COLOR_SPACE) (void *tl_camera_handle, char *output_color_space)
```

Gets the output color space of the camera.

0.3.1.1.19 TL_CAMERA_GET_CAMERA_SENSOR_TYPE

```
typedef int(* TL_CAMERA_GET_CAMERA_SENSOR_TYPE) (void *tl_camera_handle, enum TL_CAMERA_SENSOR_TYPE *camera_sensor_type)
```

Gets the camera sensor type.

0.3.1.1.20 TL_CAMERA_GET_COLOR_CORRECTION_MATRIX

```
typedef int(* TL_CAMERA_GET_COLOR_CORRECTION_MATRIX) (void *tl_camera_handle, float *matrix)
```

Gets the default color correction matrix of the camera.

0.3.1.1.21 TL_CAMERA_GET_COLOR_FILTER_ARRAY_PHASE

```
typedef int(* TL_CAMERA_GET_COLOR_FILTER_ARRAY_PHASE) (void *tl_camera_handle, enum TL_COLOR_FILTER_ARRAY_PHASE *cfaPhase)
```

Gets the the color filter array of the camera.

0.3.1.1.22 TL_CAMERA_GET_COMMUNICATION_INTERFACE

```
typedef int(* TL_CAMERA_GET_COMMUNICATION_INTERFACE) (void *tl_camera_handle, enum TL_CAMERA_COMMUNICATION_INTERFACE *communication_interface)
```

Gets the communication interface that the camera is connected to.

Parameters

tl_camera_handle	The camera handle associated with the communication interface request.
communication_interface	A reference that receives the COMMUNICATION_INTERFACE.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.23 TL_CAMERA_GET_DATA_RATE

```
typedef int(* TL_CAMERA_GET_DATA_RATE) (void *tl_camera_handle, enum TL_CAMERA_DATA_RATE *data_rate)
```

Gets the current value of the camera sensor-level data readout rate.

Parameters

tl_camera_handle	The camera handle associated with the data rate request.
data_rate	A reference that receives the current data rate value.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.24 TL_CAMERA_GET_DEFAULT_WHITE_BALANCE_MATRIX

```
typedef int(* TL_CAMERA_GET_DEFAULT_WHITE_BALANCE_MATRIX) (void *tl_camera_handle, float *matrix)
```

Gets the default white balance matrix of the camera.

0.3.1.1.25 TL_CAMERA_GET_EEP_STATUS

```
typedef int(* TL_CAMERA_GET_EEP_STATUS) (void *tl_camera_handle, enum TL_CAMERA_EEP_STATUS *eep_status_enum)
```

Equal Exposure Pulse (EEP) mode is an LVTTL-level signal that is active between the time when all rows have been reset during rolling reset, and the end of the exposure time (and the beginning of rolling readout). The signal can be used to control an external light source that will be triggered on only during the equal exposure period, providing the same amount of exposure for all pixels in the ROI.

Please see the camera documentation for details on EEP mode.

This function gets the EEP status.

Parameters

tl_camera_handle	The camera handle associated with the EEP request.
eep_status_enum	A reference that receives the current EEP status.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.26 TL_CAMERA_GET_EXPOSURE_TIME

```
typedef int(* TL_CAMERA_GET_EXPOSURE_TIME) (void *tl_camera_handle, long long *exposure_time_us)
```

Returns the current camera exposure value in microseconds.

Parameters

tl_camera_handle	The camera handle associated with the exposure request.
exposure_time_us	A reference to receive the integer exposure value in microseconds.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.27 TL_CAMERA_GET_EXPOSURE_TIME_RANGE

```
typedef int(* TL_CAMERA_GET_EXPOSURE_TIME_RANGE) (void *tl_camera_handle, long long *exposure_time_us_min, long long *exposure_time_us_max)
```

Returns the range of supported exposure values in whole microseconds for the specified camera.

Parameters

tl_camera_handle	The camera handle associated with the exposure range request.
exposure_time_us_min	A reference to receive the minimum exposure value in whole microseconds supported by the specified camera.
exposure_time_us_max	A reference to receive the maximum exposure value in whole microseconds supported by the specified camera.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.28 TL_CAMERA_GET_FIRMWARE_VERSION

```
typedef int(* TL_CAMERA_GET_FIRMWARE_VERSION) (void *tl_camera_handle, char *firmware_version, int str_length)
```

Returns a string containing the version information for all firmware components for the specified camera.

Each individual component is separated by '\r' and '
' characters with a null terminator at the end of the collection. A char buffer of size 1024 is recommended.

Parameters

tl_camera_handle	The camera handle associated with the firmware version request.
firmware_version	A pointer to a character string to receive the version information.
str_length	The length in bytes of the firmware_version buffer.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.29 TL_CAMERA_GET_FRAME_AVAILABLE_CALLBACK

```
typedef int(* TL_CAMERA_GET_FRAME_AVAILABLE_CALLBACK) (void *tl_camera_handle, TL_CAMERA_FRAME_AVAILABLE_CALLBACK *handler)
```

Gets the TL_CAMERA_FRAME_AVAILABLE_CALLBACK callback function.

Parameters

tl_camera_handle	The camera handle to associate with the callback.
handler	A pointer to the callback function. This function must conform to the TL_CAMERA_FRAME_AVAILABLE_CALLBACK prototype.

0.3.1.1.30 TL_CAMERA_GET_FRAME_RATE_CONTROL_VALUE

```
typedef int(* TL_CAMERA_GET_FRAME_RATE_CONTROL_VALUE) (void *tl_camera_handle, double *frame_rate_fps)
```

Gets the current frame rate value for the specified camera.

Parameters

tl_camera_handle	The camera handle associated with the frame rate control request.
frame_rate_fps	A reference to receive the current frame rate value in frames per second.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.31 TL_CAMERA_GET_FRAME_RATE_CONTROL_VALUE_RANGE

```
typedef int(* TL_CAMERA_GET_FRAME_RATE_CONTROL_VALUE_RANGE) (void *tl_camera_handle, double *frame_rate_fps_min, double *frame_rate_fps_max)
```

Gets the range of acceptable values for the frame rate control setting for the specified camera. If the maximum is zero then frame rate control is not supported in the camera.

Parameters

tl_camera_handle	The camera handle associated with the frame rate range request.
frame_rate_fps_max	A reference to receive the maximum frame rate in frames per second.
frame_rate_fps_min	A reference to receive the minimum frame rate in frames per second.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.32 TL_CAMERA_GET_FRAME_TIME

```
typedef int(* TL_CAMERA_GET_FRAME_TIME) (void *tl_camera_handle, int *frame_time_us)
```

Returns the time, in microseconds (us), required for a frame to be exposed and read out from the sensor. When triggering frames, this property may be used to determine when the camera is ready to accept another trigger. Other factors such as the communication speed between the camera and the host computer can affect the maximum trigger rate.

NOTE: This parameters depends on tl_camera_set_frames_per_trigger_zero_for_unlimited and tl_camera_set_exposure_time.

Parameters

tl_camera_handle	The camera handle associated with the frame time request.
frame_time_us	A reference to receive the current frame_time_us value for the specified camera.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.33 TL_CAMERA_GET_FRAMES_PER_TRIGGER_RANGE

```
typedef int(* TL_CAMERA_GET_FRAMES_PER_TRIGGER_RANGE) (void *tl_camera_handle, unsigned int *number_of_frames_per_trigger_min, unsigned int *number↩
_of_frames_per_trigger_max)
```

Gets the range of acceptable values for the frames per trigger camera parameter.

Parameters

tl_camera_handle	The camera handle associated with the frames per trigger request.
number_of_frames_per_trigger_min	A reference that receives the minimum valid frames per trigger value.
number_of_frames_per_trigger_max	A reference that receives the maximum valid frames per trigger value.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.34 TL_CAMERA_GET_FRAMES_PER_TRIGGER_ZERO_FOR_UNLIMITED

```
typedef int(* TL_CAMERA_GET_FRAMES_PER_TRIGGER_ZERO_FOR_UNLIMITED) (void *tl_camera_handle, unsigned int *number_of_frames_per_trigger_or_zero_↩
for_unlimited)
```

Gets the number of frames per trigger.

Parameters

tl_camera_handle	The camera handle associated with the frames per trigger request.
number_of_frames_per_trigger_or_zero_for_unlimited	A reference that receives the number of frames per trigger value.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.35 TL_CAMERA_GET_GAIN

```
typedef int(* TL_CAMERA_GET_GAIN) (void *tl_camera_handle, int *gain)
```

Gets the current gain value for the specified camera.

The units of measure for this value vary by camera model. To convert this value to decibels (dB), use tl_camera_convert_gain_to_decibels().

Parameters

tl_camera_handle	The camera handle associated with the gain request.
gain	A reference to receive the current gain value.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.36 TL_CAMERA_GET_GAIN_RANGE

```
typedef int(* TL_CAMERA_GET_GAIN_RANGE) (void *tl_camera_handle, int *gain_min, int *gain_max)
```

Get the range of possible gain values.

The units of measure for this value vary by camera model. To convert this value to decibels (dB), use tl_camera_convert_gain_to_decibels().

Parameters

tl_camera_handle	The camera handle associated with the image width range request.
gain_min	A reference that receives the minimum value in dB $*$ 10 for gain.
gain_max	A reference that receives the maximum value in dB $*$ 10 for gain.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.37 TL_CAMERA_GET_HOT_PIXEL_CORRECTION_THRESHOLD

```
typedef int(* TL_CAMERA_GET_HOT_PIXEL_CORRECTION_THRESHOLD) (void *tl_camera_handle, int *hot_pixel_correction_threshold)
```

This function may be used to get the current threshold value for hot-pixel correction.

This value is a quantitative measure of how aggressively the camera will remove hot pixels.

Parameters

tl_camera_handle	The camera handle associated with the hot pixel correction threshold command.
hot_pixel_correction_threshold	A reference that receives the current value of the hot pixel correction threshold for the specified camera.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.38 TL_CAMERA_GET_HOT_PIXEL_CORRECTION_THRESHOLD_RANGE

```
typedef int(* TL_CAMERA_GET_HOT_PIXEL_CORRECTION_THRESHOLD_RANGE) (void *tl_camera_handle, int *hot_pixel_correction_threshold_min, int *hot_↩
pixel_correction_threshold_max)
```

This function may be used to get the range of acceptable hot pixel correction threshold values.

If the maximum value is 0 (zero), that is an indication that hot pixel correction is not supported by the specified camera.

Parameters

tl_camera_handle	The camera handle associated with the hot pixel correction threshold command.
hot_pixel_correction_threshold_min	A reference that receives the minimum acceptable value for the hot pixel correction threshold.
hot_pixel_correction_threshold_max	A reference that receives the maximum acceptable value for the hot pixel correction threshold.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.39 TL_CAMERA_GET_IMAGE_HEIGHT

```
typedef int(* TL_CAMERA_GET_IMAGE_HEIGHT) (void *tl_camera_handle, int *height_pixels)
```

Gets the current image height in pixels.

Parameters

tl_camera_handle	The camera handle associated with the image width request.
height_pixels	A reference that receives the value in pixels for the current image height.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.40 TL_CAMERA_GET_IMAGE_HEIGHT_RANGE

```
typedef int(* TL_CAMERA_GET_IMAGE_HEIGHT_RANGE) (void *tl_camera_handle, int *image_height_pixels_min, int *image_height_pixels_max)
```

Gets the range of possible image height values.

Parameters

tl_camera_handle	The camera handle associated with the image height range request.
image_height_pixels_min	A reference that receives the minimum value in pixels for the image height.
image_height_pixels_max	A reference that receives the maximum value in pixels for the image height.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.41 TL_CAMERA_GET_IMAGE_POLL_TIMEOUT

```
typedef int(* TL_CAMERA_GET_IMAGE_POLL_TIMEOUT) (void *tl_camera_handle, int *timeout_ms)
```

Returns the current camera image poll time out value in milliseconds.

Parameters

tl_camera_handle	The camera handle associated with the exposure request.
timeout_ms	A reference to receive the time out value.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.42 TL_CAMERA_GET_IMAGE_WIDTH

```
typedef int(* TL_CAMERA_GET_IMAGE_WIDTH) (void *tl_camera_handle, int *width_pixels)
```

Gets the current image width in pixels.

Parameters

tl_camera_handle	The camera handle associated with the image width request.
width_pixels	A reference that receives the value in pixels for the current image width.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.43 TL_CAMERA_GET_IMAGE_WIDTH_RANGE

```
typedef int(* TL_CAMERA_GET_IMAGE_WIDTH_RANGE) (void *tl_camera_handle, int *image_width_pixels_min, int *image_width_pixels_max)
```

Get the range of possible image width values.

Parameters

tl_camera_handle	The camera handle associated with the image width range request.
image_width_pixels_min	A reference that receives the minimum value in pixels for the image width.
image_width_pixels_max	A reference that receives the maximum value in pixels for the image width.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.44 TL_CAMERA_GET_IS_ARMED

```
typedef int(* TL_CAMERA_GET_IS_ARMED) (void *tl_camera_handle, int *is_armed)
```

Gets the camera is armed or not.

0.3.1.1.45 TL_CAMERA_GET_IS_COOLING_ENABLED

```
typedef int(* TL_CAMERA_GET_IS_COOLING_ENABLED) (void *tl_camera_handle, int *is_cooling_enabled)
```

Determine if active-cooling mode is enabled. Some camera models include special hardware that provides additional cooling (beyond the conventional passive cooling hardware) for the sensor and the internal camera chamber.
To determine if a camera supports active cooling, use tl_camera_get_is_cooling_supported().

Parameters

tl_camera_handle	The camera handle associated with the cooling mode get request.
is_cooling_enabled	A reference that receives the cooling mode enable status. 0 (zero) for off and 1 (one) for on.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.46 TL_CAMERA_GET_IS_COOLING_SUPPORTED

```
typedef int(* TL_CAMERA_GET_IS_COOLING_SUPPORTED) (void *tl_camera_handle, int *is_cooling_supported)
```

Gets the cooling is supported or not of the camera.

0.3.1.1.47 TL_CAMERA_GET_IS_DATA_RATE_SUPPORTED

```
typedef int(* TL_CAMERA_GET_IS_DATA_RATE_SUPPORTED) (void *tl_camera_handle, enum TL_CAMERA_DATA_RATE data_rate, int *is_data_rate_supported)
```

Scientific-CCD cameras and compact-scientific cameras handle sensor- level data-readout speed differently.

Use this method to test whether the connected camera supports a particular data rate.

Parameters

tl_camera_handle	The camera handle associated with the data rate request.
data_rate	The data rate enumeration value to check for support.
is_data_rate_supported	An indication of whether or not the data rate value is supported. A 1 (one) indicates that the specified data rate is supported and a 0 (zero) indicates that the specified data rate is not supported.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.48 TL_CAMERA_GET_IS_EEP_SUPPORTED

```
typedef int(* TL_CAMERA_GET_IS_EEP_SUPPORTED) (void *tl_camera_handle, int *is_eep_supported)
```

Gets the EEP is supported or not of the camera.

0.3.1.1.49 TL_CAMERA_GET_IS_FRAME_RATE_CONTROL_ENABLED

```
typedef int(* TL_CAMERA_GET_IS_FRAME_RATE_CONTROL_ENABLED) (void *tl_camera_handle, int *is_enabled)
```

Gets the status of the frame rate control.

Parameters

tl_camera_handle	The camera handle associated with the frame rate control request.
is_enabled	A value that returns the current frame rate control status.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.50 TL_CAMERA_GET_IS_HOT_PIXEL_CORRECTION_ENABLED

```
typedef int(* TL_CAMERA_GET_IS_HOT_PIXEL_CORRECTION_ENABLED) (void *tl_camera_handle, int *is_hot_pixel_correction_enabled)
```

Due to variability in manufacturing, some pixels have inherently higher dark current which manifests as abnormally bright pixels in images, typically visible with longer exposures. Hot-pixel correction identifies hot pixels and then substitutes a calculated value based on the values of neighboring pixels in place of hot pixels.

This function may be used to get the current state of hot-pixel correction.

Parameters

tl_camera_handle	The camera handle associated with the hot pixel correction request.
is_hot_pixel_correction_enabled	A reference that receives the state of the hot pixel correction functionality for the specified camera. A 0 (zero) value indicates that hot pixel correction is disabled and a 1 (one) indicates that it is enabled.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.51 TL_CAMERA_GET_IS_LED_ON

```
typedef int(* TL_CAMERA_GET_IS_LED_ON) (void *tl_camera_handle, int *is_led_on)
```

Some scientific cameras include an LED indicator light on the back panel.

This function gets the LED status.

Parameters

tl_camera_handle	The camera handle associated with the LED request.
is_led_on	A reference that receives the LED status. 0 (zero) for off and 1 (one) for on.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.52 TL_CAMERA_GET_IS_LED_SUPPORTED

```
typedef int(* TL_CAMERA_GET_IS_LED_SUPPORTED) (void *tl_camera_handle, int *is_led_supported)
```

Gets the LED is supported or not of the camera.

0.3.1.1.53 TL_CAMERA_GET_IS_NIR_BOOST_SUPPORTED

```
typedef int(* TL_CAMERA_GET_IS_NIR_BOOST_SUPPORTED) (void *tl_camera_handle, int *is_nir_boost_supported)
```

Gets the NIRBoost is supported or not of the camera.

0.3.1.1.54 TL_CAMERA_GET_IS_OPERATION_MODE_SUPPORTED

```
typedef int(* TL_CAMERA_GET_IS_OPERATION_MODE_SUPPORTED) (void *tl_camera_handle, enum TL_CAMERA_OPERATION_MODE operation_mode, int *is_operation↩
_mode_supported)
```

Gets the operation mode of the camera.

0.3.1.1.55 TL_CAMERA_GET_IS_TAPS_SUPPORTED

```
typedef int(* TL_CAMERA_GET_IS_TAPS_SUPPORTED) (void *tl_camera_handle, int *is_taps_supported, enum TL_CAMERA_TAPS tap)
```

Gets the tap is supported or not of the camera.

0.3.1.1.56 TL_CAMERA_GET_LAST_ERROR

```
typedef char*(* TL_CAMERA_GET_LAST_ERROR) (void)
```

Provides a character string that describes the most recent error that has occurred.

Returns

A character string indicating the most recent error that has occurred. The application must NOT attempt modify or deallocate the character string. Any attempt to do so is not permitted and could result in undefined behavior.

0.3.1.1.57 TL_CAMERA_GET_MEASURED_FRAME_RATE

```
typedef int(* TL_CAMERA_GET_MEASURED_FRAME_RATE) (void *tl_camera_handle, double *frames_per_second)
```

Gets the current rate of frames in frames per second that are delivered to the host computer. The frame rate can be affected by the performance capabilities of the host computer and the communication interface.

This method can be polled for updated values as needed.

Parameters

tl_camera_handle	The camera handle associated with the fps request.
frames_per_second	A reference to receive the current fps value for the specified camera. Note that this parameter is specified as a reference to a double precision floating point value.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.58 TL_CAMERA_GET_MODEL

```
typedef int(* TL_CAMERA_GET_MODEL) (void *tl_camera_handle, char *model, int str_length)
```

Gets the camera model information.

Parameters

tl_camera_handle	The camera handle associated with the model info request.
model	A pointer to a character buffer to receive the model information.
str_length	The length in bytes of the specified character buffer.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.59 TL_CAMERA_GET_MODEL_STRING_LENGTH_RANGE

```
typedef int(* TL_CAMERA_GET_MODEL_STRING_LENGTH_RANGE) (void *tl_camera_handle, int *model_min, int *model_max)
```

Gets the range of valid character string buffer lengths that must be specified to receive the camera model string.

Parameters

tl_camera_handle	The camera handle associated with the model info request.
model_min	A reference that receives the minimum length of the model character string.
model_max	A reference that receives the maximum length of the model character string.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.60 TL_CAMERA_GET_NAME

```
typedef int(* TL_CAMERA_GET_NAME) (void *tl_camera_handle, char *name, int str_length)
```

Gets the camera name.

Parameters

tl_camera_handle	The camera handle associated with the name request.
name	A pointer to a character string to receive the camera name.
str_length	The length of the name character string.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.61 TL_CAMERA_GET_NAME_STRING_LENGTH_RANGE

```
typedef int(* TL_CAMERA_GET_NAME_STRING_LENGTH_RANGE) (void *tl_camera_handle, int *name_min, int *name_max)
```

Gets the range of valid character string buffer lengths that must be specified to receive or set the camera name string.

Parameters

tl_camera_handle	The camera handle associated with the name request.
name_min	A reference that receives the minimum length of the name character string.
name_max	A reference that receives the maximum length of the name character string.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.62 TL_CAMERA_GET_NIR_BOOST_ENABLE

```
typedef int(* TL_CAMERA_GET_NIR_BOOST_ENABLE) (void *tl_camera_handle, int *nir_boost_enable)
```

Determine if near-infrared-boost mode is enabled. Some camera models include support for boosting the intensity of wavelengths of light in the near-infrared part of the spectrum. To determine if a camera supports NIR-boost mode, use tl_camera_get_is_nir_boost_supported().

Parameters

tl_camera_handle	The camera handle associated with the NIR boost get request.
nir_boost_enable	A reference that receives the NIR boost enable status. 0 (zero) for off and 1 (one) for on.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.63 TL_CAMERA_GET_OPERATION_MODE

```
typedef int(* TL_CAMERA_GET_OPERATION_MODE) (void *tl_camera_handle, enum TL_CAMERA_OPERATION_MODE *operation_mode)
```

Gets the operation mode of the camera.

0.3.1.1.64 TL_CAMERA_GET_PENDING_FRAME_OR_NULL

```
typedef int(* TL_CAMERA_GET_PENDING_FRAME_OR_NULL) (void *tl_camera_handle, unsigned short **image_buffer, int *frame_count, unsigned char **metadata,
int *metadata_size_in_bytes)
```

Returns the current camera frame or null.

IMPORTANT: The memory is allocated inside the camera SDK. Therefore, the provided pointer will be set to a position in the internal buffer. Either complete all tasks related to this image or make a copy of the data before returning from this function. Once this function exits, the pointer to the image buffer becomes invalid.

NOTE: There are two methods for getting image frames from the camera: Polling or registering for a callback.

1. Poll with the tl_camera_get_pending_frame_or_null function, typically from the main thread (polling from any thread is valid).

2. Register for a callback. In this case, frames will arrive on a worker thread to avoid interrupting the main thread. Be sure to use proper thread-locking techniques if the data needs to be marshaled from the worker thread to the main thread (such as for display in a graphical user interface).

Parameters

tl_camera_handle	The camera handle associated with the image request.
image_buffer	The pointer to the buffer that contains the image data. The byte ordering of the data in this buffer is little-endian. IMPORTANT: This pointer is only valid for the duration of this callback.

The data for both color and monochrome cameras are ordered left to right across a row followed by the next row below it.

For monochrome cameras, each pixel requires two bytes.

For color cameras, each pixel requires two bytes for blue followed by two bytes for green followed by two bytes for red (BBGGRRBBGGRRBBGGRR...). Therefore, each color pixel requires six bytes.

Parameters

frame_count	The image count corresponding to the received image during the current acquisition run. If the image metadata section was not found, this will be 0.
metadata	The pointer to the buffer that contains the image metadata. The byte ordering of the data in this buffer is little-endian. If the metadata section was not found, this will be null. IMPORTANT: This pointer is only valid for the duration of this callback.
metadata_size_in_bytes	The size (in bytes) of the image metadata buffer. If the metadata section was not found, this will be 0.

Returns

0 if successful or a positive integer error code to indicate failure.

The metadata section consists of tag ID/value pairs in the following format:

|---— Tag ID (4 ASCII characters [4 bytes]) ---—|---— Tag data [4 bytes] ---—|

These pairs occur consecutively in memory for all the tags supported by the camera.

To iterate over the tag/value pairs, start with the pointer to the metadata and increment by 8 bytes to go from one pair to the next. Either the size of the metadata buffer or the detection of the ENDT tag should be used to determine when to stop.

The following table lists the metadata tags along with a brief description:

Table 51 Image Metadata Tag Description

Tag ID	Description
TSI\0	Start of metadata tag region - the data value of this tag is always 0
FCNT	Frame count
PCKH	Pixel clock count - upper 32 bits
PCKL	Pixel clock count - lower 32 bits
IFMT	Image data format
IOFF	Offset to pixel data in multiple of 8 bytes
ENDT	End of metadata tag region - the data value of this tag is always 0

NOTE: Not all tags are supported (are present in the metadata tag section) by all camera models. The IFMT (Image data format) tag is not present on the Zelux or Quantalux cameras.

How-To: Read Tags

The example below assumes `tl_camera_get_pending_frame_or_null()` will be called and `metadata` is not NULL. It shows one way of taking the metadata pointer and iterating over the tag / value pairs.

```
unsigned char* metadata = 0;
int metadata_size_in_bytes = 0;
// call get_pending_frame_or_null()
if (metadata)
{
    int metadata_index = 0; // index (in bytes) into metadata
    while (metadata_index < metadata_size_in_bytes)
    {
        unsigned char* metadata_entry = metadata + metadata_index; // byte pointer to the tag / value pair
        unsigned long value = *(unsigned long*)(metadata_entry + 4); // the value is the second 4 bytes in the tag / value pair
        if (strncmp(metadata_entry, "ENDT", 4))// only want to compare first 4 bytes
        {
            break; // this is the terminating tag
        }
        else if (strncmp(metadata_entry, "FCNT", 4))
        {
            // etc...
        }
        // check for any desired tags and operate on value accordingly
        metadata_index += 8; // there are  8 bytes per tag / value pair
    }
}
```

How-To: Calculate Relative Timestamp (nanoseconds)

Pixel clock count can be used to get a relative timestamp using the following formula: `relative_timestamp_ns = (pixel_clock_count / timestamp_clock_frequency) * 1000000000`

- `pixel_clock_count` can be found by combining the upper and lower pixel clock counts from the metadata: `(pixel_clock_count_upper << 32) | pixel_clock_count_lower`.
- `timestamp_clock_frequency` can be found using the following function: `tl_camera_get_timestamp_clock_frequency()`.

This timestamp is relative to an internal timer and is recorded immediately after exposure finishes. To get the time elapsed, subtract the relative timestamp of the first frame from the relative timestamp of the last frame. If `tl_camera_get_timestamp_clock_frequency()` returns an error code, then the camera does not support relative time stamping.

0.3.1.1.65 TL_CAMERA_GET_POLAR_PHASE

```
typedef int(* TL_CAMERA_GET_POLAR_PHASE) (void *tl_camera_handle, enum TL_POLARIZATION_PROCESSOR_POLAR_PHASE *polar_phase)
```

Gets the camera polar phase information.

Parameters

tl_camera_handle	The camera handle associated with the model info request.
polar_phase	A pointer to a TL_POLARIZATION_PROCESSOR_POLAR_PHASE enumeration value.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.66 TL_CAMERA_GET_ROI

```
typedef int(* TL_CAMERA_GET_ROI) (void *tl_camera_handle, int *upper_left_x_pixels, int *upper_left_y_pixels, int *lower_right_x_pixels, int *lower↩
_right_y_pixels)
```

Gets the current Region of Interest (ROI) values.

The ROI is specified by 2 sets of x,y coordinates that establish a bounded rectangular area:

- 1 for the upper left corner
- 1 for the lower right corner.

Parameters

tl_camera_handle	The camera handle associated with the ROI request.
upper_left_x_pixels	A reference to receive the x coordinate of the upper left corner of the ROI.
upper_left_y_pixels	A reference to receive the y coordinate of the upper left corner of the ROI.
lower_right_x_pixels	A reference to receive the x coordinate of the lower right corner of the ROI.
lower_right_y_pixels	A reference to receive the y coordinate of the lower right corner of the ROI.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.67 TL_CAMERA_GET_ROI_RANGE

```
typedef int(* TL_CAMERA_GET_ROI_RANGE) (void *tl_camera_handle, int *upper_left_x_pixels_min, int *upper_left_y_pixels_min, int *lower_right_x_↩
pixels_min, int *lower_right_y_pixels_min, int *upper_left_x_pixels_max, int *upper_left_y_pixels_max, int *lower_right_x_pixels_max, int *lower_↩
right_y_pixels_max)
```

Gets the range of acceptable values for Region of Interest (ROI) coordinates.

The ROI is specified by 2 sets of x,y coordinates that establish a bounded rectangular area:

- 1 for the upper left corner
- 1 for the lower right corner.

Parameters

tl_camera_handle	The camera handle associated with the ROI range request.
upper_left_x_pixels_min	A reference to receive the the minimum x coordinate of the upper left corner of the ROI.
upper_left_y_pixels_min	A reference to receive the the minimum y coordinate of the upper left corner of the ROI.
lower_right_x_pixels_min	A reference to receive the the minimum x coordinate of the lower right corner of the ROI.
lower_right_y_pixels_min	A reference to receive the the minimum y coordinate of the lower right corner of the ROI.
upper_left_x_pixels_max	A reference to receive the the maximum x coordinate of the upper left corner of the ROI.
upper_left_y_pixels_max	A reference to receive the the maximum y coordinate of the upper left corner of the ROI.
lower_right_x_pixels_max	A reference to receive the the maximum x coordinate of the lower right corner of the ROI.
lower_right_y_pixels_max	A reference to receive the the maximum y coordinate of the lower right corner of the ROI.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.68 TL_CAMERA_GET_SENSOR_HEIGHT

```
typedef int(* TL_CAMERA_GET_SENSOR_HEIGHT) (void *tl_camera_handle, int *height_pixels)
```

Gets the sensor height in pixels.

Parameters

tl_camera_handle	The camera handle associated with the sensor height request.
height_pixels	A reference that receives the value in pixels for the sensor height.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.69 TL_CAMERA_GET_SENSOR_PIXEL_HEIGHT

```
typedef int(* TL_CAMERA_GET_SENSOR_PIXEL_HEIGHT) (void *tl_camera_handle, double *pixel_height_um)
```

Get the physical height, in micrometers, of a single light-sensitive photo site on the sensor.

Parameters

tl_camera_handle	The camera handle associated with the pixel height request.
pixel_height_um	A reference to receive the current pixel height value in micrometers.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.70 TL_CAMERA_GET_SENSOR_PIXEL_SIZE_BYTES

```
typedef int(* TL_CAMERA_GET_SENSOR_PIXEL_SIZE_BYTES) (void *tl_camera_handle, int *sensor_pixel_size_bytes)
```

Get the current pixel size in bytes. This represents the amount of space 1 pixel will occupy in the frame buffer.

Parameters

tl_camera_handle	The camera handle associated with the pixel size request.
sensor_pixel_size_bytes	A reference to receive the current pixel size value in bytes.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.71 TL_CAMERA_GET_SENSOR_PIXEL_WIDTH

```
typedef int(* TL_CAMERA_GET_SENSOR_PIXEL_WIDTH) (void *tl_camera_handle, double *pixel_width_um)
```

Get the physical width, in micrometers, of a single light-sensitive photo site on the sensor.

Parameters

tl_camera_handle	The camera handle associated with the pixel width request.
pixel_width_um	A reference to receive the current pixel width value in micrometers.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.72 TL_CAMERA_GET_SENSOR_READOUT_TIME

```
typedef int(* TL_CAMERA_GET_SENSOR_READOUT_TIME) (void *tl_camera_handle, int *sensor_readout_time_ns)
```

Gets the send readout time for the specified camera.

Parameters

tl_camera_handle	The camera handle associated with the sensor readout time request.
sensor_readout_time_ns	A reference to receive the current send readout time value in nanoseconds.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.73 TL_CAMERA_GET_SENSOR_WIDTH

```
typedef int(* TL_CAMERA_GET_SENSOR_WIDTH) (void *tl_camera_handle, int *width_pixels)
```

Gets the sensor width in pixels.

Parameters

tl_camera_handle	The camera handle associated with the sensor width request.
width_pixels	A reference that receives the value in pixels for the sensor width.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.74 TL_CAMERA_GET_SERIAL_NUMBER

```
typedef int(* TL_CAMERA_GET_SERIAL_NUMBER) (void *tl_camera_handle, char *serial_number, int str_length)
```

Gets the camera serial number.

Parameters

tl_camera_handle	The camera handle associated with the serial number request.
serial_number	A pointer to a character string to receive the camera serial number.
str_length	The length of the serial number character string.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.75 TL_CAMERA_GET_SERIAL_NUMBER_STRING_LENGTH_RANGE

```
typedef int(* TL_CAMERA_GET_SERIAL_NUMBER_STRING_LENGTH_RANGE) (void *tl_camera_handle, int *serial_number_min, int *serial_number_max)
```

Gets the range of valid character string buffer lengths that must be specified to receive the camera serial number string.

Parameters

tl_camera_handle	The camera handle associated with the serial number range request.
serial_number_min	A reference that receives the minimum length of the serial number character string.
serial_number_max	A reference that receives the maximum length of the serial number character string.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.76 TL_CAMERA_GET_TAP_BALANCE_ENABLE

```
typedef int(* TL_CAMERA_GET_TAP_BALANCE_ENABLE) (void *tl_camera_handle, int *taps_balance_enable)
```

Gets the value of the current camera tap balance setting. The higher frame rates enabled by multi-tap operation are not without tradeoffs. Since each tap has a different analog to digital converter with a different gain, this difference can manifest in the image by each half (or quadrant) having slightly different intensities. The tap balance feature mitigates this effect across a wide range of exposure, gain, and black level values.

Parameters

tl_camera_handle	The camera handle associated with the taps request.
taps_balance_enable	A reference that receives the tap balance enable status. 0 (zero) for off and 1 (one) for on.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.77 TL_CAMERA_GET_TAPS

```
typedef int(* TL_CAMERA_GET_TAPS) (void *tl_camera_handle, enum TL_CAMERA_TAPS *taps)
```

Gets the current camera taps value. Scientific CCD cameras support one or more taps.

After exposure is complete, a CCD pixel array holds the charge corresponding to the amount of light collected at each pixel location. The data is then read out through 1, 2, or 4 channels at a time. Reading the data through more than 1 channel (for cameras that support multi-tap operation) can enable higher maximum frame rates.

Parameters

tl_camera_handle	The camera handle associated with the taps request.
taps	A pointer to a TL_CAMERA_TAPS enumeration value.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.78 TL_CAMERA_GET_TIMESTAMP_CLOCK_FREQUENCY

```
typedef int(* TL_CAMERA_GET_TIMESTAMP_CLOCK_FREQUENCY) (void *tl_camera_handle, int *timestamp_clock_frequency_hz_or_zero)
```

Gets the timestamp clock frequency for the camera in Hz. This can be used along with the clock count value in each frame's metadata to calculate the relative time from plug for that frame.

Parameters

tl_camera_handle	The camera handle associated with the timestamp clock frequency request.
timestamp_clock_frequency_hz_or_zero	A reference to receive the current time stamp clock frequency value in Hz or zero if unsupported.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.79 TL_CAMERA_GET_TRIGGER_POLARITY

```
typedef int(* TL_CAMERA_GET_TRIGGER_POLARITY) (void *tl_camera_handle, enum TL_CAMERA_TRIGGER_POLARITY *trigger_polarity_enum)
```

Gets the current hardware trigger polarity of the specified camera.

Parameters

tl_camera_handle	The camera handle associated with the hardware trigger mode request.
trigger_polarity_enum	A reference to a TRIGGER_POLARITY to receive the currently configured trigger polarity.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.80 TL_CAMERA_GET_USB_PORT_TYPE

```
typedef int(* TL_CAMERA_GET_USB_PORT_TYPE) (void *tl_camera_handle, enum TL_CAMERA_USB_PORT_TYPE *usb_port_type)
```

Gets the USB port type that the camera is connected to.

Parameters

tl_camera_handle	The camera handle associated with the USB port type request.
usb_port_type	A reference that receives the USB port type.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.81 TL_CAMERA_GET_USER_MEMORY

```
typedef int(* TL_CAMERA_GET_USER_MEMORY) (void *tl_camera_handle, unsigned char *destination_data_buffer, long long number_of_bytes_to_read, long
long camera_user_memory_offset_bytes)
```

Read on-camera, non-volatile memory that is available to the user.

Use tl_camera_get_user_memory_maximum_size to query the available memory.

Parameters

tl_camera_handle	The camera handle associated with the USB port type request.
destination_data_buffer	A byte buffer in which to receive the specified number of bytes.
number_of_bytes_to_read	The number of bytes of user memory to read.
camera_user_memory_offset_bytes	A byte offset in the on-camera, non-volatile memory from which to start reading. Use a value of zero to read from the beginning of user memory.

Returns

0 if successful or a positive integer error code to indicate failure.

See also

TL_CAMERA_GET_USER_MEMORY_MAXIMUM_SIZE, TL_CAMERA_SET_USER_MEMORY

0.3.1.1.82 TL_CAMERA_GET_USER_MEMORY_MAXIMUM_SIZE

```
typedef int(* TL_CAMERA_GET_USER_MEMORY_MAXIMUM_SIZE) (void *tl_camera_handle, long long *maximum_size_bytes)
```

Gets the number of bytes of on-camera, non-volatile memory storage available to the user.

Parameters

tl_camera_handle	The camera handle associated with the USB port type request.
maximum_size_bytes	A reference that receives the 64-bit number of available bytes.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.83 TL_CAMERA_ISSUE_SOFTWARE_TRIGGER

```
typedef int(* TL_CAMERA_ISSUE_SOFTWARE_TRIGGER) (void *tl_camera_handle)
```

This function will generate a trigger through the camera SDK rather than through the hardware trigger input.

The behavior of a software trigger depends on the number of frames configured using tl_camera_set_number_of_frames_per_trigger.

- If the number of frames per trigger is set to zero, then a single software trigger will start continuous-video mode.

- If the number of frames per trigger is set to one or higher, then one software trigger will generate the requested number of frames. In this case, it is important to avoid issuing subsequent software triggers until the time specified by tl_camera_get_frame_time multiplied by the number of frames has elapsed. If insufficient time elapses, the trigger is ignored by the camera.
  NOTE: Some versions of camera firmware exhibit a longer frame time than is ideal. Please check the website for the latest available firmware.

  Multiple software triggers can be issued before calling Disarm().

  See also tl_camera_get_frame_time, tl_camera_get_exposure_time, and tl_camera_get_sensor_readout_time.

Parameters

tl_camera_handle	The camera handle for issuing a software trigger.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.84 TL_CAMERA_OPEN_CAMERA

```
typedef int(* TL_CAMERA_OPEN_CAMERA) (char *camera_serial_number, void **tl_camera_handle)
```

Gets a handle to a camera with the specified serial number.

The handle represents the software abstraction of the physical camera.

The returned handle must be used with most API functions to perform the camera specific task corresponding to the particular function.

Parameters

camera_serial_number	The camera serial number.
tl_camera_handle	A reference to receive the handle to the camera with the specified serial number.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.85 TL_CAMERA_OPEN_SDK

```
typedef int(* TL_CAMERA_OPEN_SDK) (void)
```

The tl_camera_open_sdk function is used to initialize the SDK. This function must be called prior to calling any other API function.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.86 TL_CAMERA_SET_BINX

```
typedef int(* TL_CAMERA_SET_BINX) (void *tl_camera_handle, int binx)
```

Binning sums adjacent sensor pixels into "super pixels". It trades off spatial resolution for sensitivity and speed. For example, if a sensor is 1920 by 1080 pixels and binning is set to two in the X direction and two in the Y direction, the resulting image will be 960 by 540 pixels. Since smaller images require less data to be transmitted to the host computer, binning may increase the frame rate. By default, binning is set to one in both horizontal and vertical directions.

Sets the current horizontal binning value for the specified camera.

Parameters

tl_camera_handle	The camera handle associated with the horizontal binning command.
binx	A value that is used to configure the horizontal binning setting.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.87 TL_CAMERA_SET_BINY

```
typedef int(* TL_CAMERA_SET_BINY) (void *tl_camera_handle, int biny)
```

Binning sums adjacent sensor pixels into "super pixels". It trades off spatial resolution for sensitivity and speed. For example, if a sensor is 1920 by 1080 pixels and binning is set to two in the X direction and two in the Y direction, the resulting image will be 960 by 540 pixels. Since smaller images require less data to be transmitted to the host computer, binning may increase the frame rate. By default, binning is set to one in both horizontal and vertical directions.

Sets the current vertical binning value for the specified camera.

Parameters

tl_camera_handle	The camera handle associated with the vertical binning command.
biny	A value that is used to configure the vertical binning setting.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.88 TL_CAMERA_SET_BLACK_LEVEL

```
typedef int(* TL_CAMERA_SET_BLACK_LEVEL) (void *tl_camera_handle, int black_level)
```

Sets the black level value for the specified camera.

Parameters

tl_camera_handle	The camera handle associated with the black level command.
black_level	A value that is used to configure the black level setting.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.89 TL_CAMERA_SET_CAMERA_CONNECT_CALLBACK

```
typedef int(* TL_CAMERA_SET_CAMERA_CONNECT_CALLBACK) (TL_CAMERA_CONNECT_CALLBACK handler, void *context)
```

Sets the TL_CAMERA_CONNECT_CALLBACK callback function.

Parameters

handler	A pointer to the callback function. This function must conform to the TL_CAMERA_CONNECT_CALLBACK prototype.
context	A pointer to a user specified context. This parameter is ignored by the SDK.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.90 TL_CAMERA_SET_CAMERA_DISCONNECT_CALLBACK

```
typedef int(* TL_CAMERA_SET_CAMERA_DISCONNECT_CALLBACK) (TL_CAMERA_DISCONNECT_CALLBACK handler, void *context)
```

Sets the TL_CAMERA_DISCONNECT_CALLBACK callback function.

Parameters

handler	A pointer to the callback function. This function must conform to the TL_CAMERA_DISCONNECT_CALLBACK prototype.
context	A pointer to a user specified context. This parameter is ignored by the SDK.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.91 TL_CAMERA_SET_DATA_RATE

```
typedef int(* TL_CAMERA_SET_DATA_RATE) (void *tl_camera_handle, enum TL_CAMERA_DATA_RATE data_rate)
```

Sets the current value of the camera sensor-level data readout rate.

Parameters

tl_camera_handle	The camera handle associated with the data rate request.
data_rate	The data rate value to set.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.92 TL_CAMERA_SET_EXPOSURE_TIME

```
typedef int(* TL_CAMERA_SET_EXPOSURE_TIME) (void *tl_camera_handle, long long exposure_time_us)
```

Sets the specified camera's exposure to a value which must be specified in microseconds.

Parameters

tl_camera_handle	The camera handle associated with the exposure change.
exposure_time_us	A whole number of microseconds which represents the new exposure value.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.93 TL_CAMERA_SET_FRAME_AVAILABLE_CALLBACK

```
typedef int(* TL_CAMERA_SET_FRAME_AVAILABLE_CALLBACK) (void *tl_camera_handle, TL_CAMERA_FRAME_AVAILABLE_CALLBACK handler, void *context)
```

Sets the TL_CAMERA_FRAME_AVAILABLE_CALLBACK callback function.

NOTE: There are two methods for getting image frames from the camera: Polling or registering for a callback.

1. Poll with the tl_camera_get_pending_frame_or_null function, typically from the main thread (polling from any thread is valid).

2. Register for a callback. In this case, frames will arrive on a worker thread to avoid interrupting the main thread. Be sure to use proper thread-locking techniques if the data needs to be marshaled from the worker thread to the main thread (such as for display in a graphical user interface).

For details, please see the documentation for TL_CAMERA_FRAME_AVAILABLE_CALLBACK.

Parameters

tl_camera_handle	The camera handle to associate with the callback.
handler	A pointer to the callback function. This function must conform to the TL_CAMERA_FRAME_AVAILABLE_CALLBACK prototype.
context	A pointer to a user specified context. This parameter is ignored by the SDK.

0.3.1.1.94 TL_CAMERA_SET_FRAME_RATE_CONTROL_VALUE

```
typedef int(* TL_CAMERA_SET_FRAME_RATE_CONTROL_VALUE) (void *tl_camera_handle, double frame_rate_fps)
```

Sets the frame rate value in frames per second for the specified camera.

Parameters

tl_camera_handle	The camera handle associated with the frame rate control request.
frame_rate_fps	A value that is used to configure the frame rate setting in frames per second.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.95 TL_CAMERA_SET_FRAMES_PER_TRIGGER_ZERO_FOR_UNLIMITED

```
typedef int(* TL_CAMERA_SET_FRAMES_PER_TRIGGER_ZERO_FOR_UNLIMITED) (void *tl_camera_handle, unsigned int number_of_frames_per_trigger_or_zero_↩
for_unlimited)
```

Sets the number of frames per trigger.

Parameters

tl_camera_handle	The camera handle associated with the frames per trigger request.
number_of_frames_per_trigger_or_zero_for_unlimited	The number of frames per trigger value to set.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.96 TL_CAMERA_SET_GAIN

```
typedef int(* TL_CAMERA_SET_GAIN) (void *tl_camera_handle, int gain)
```

Sets the current gain value for the specified camera.

Parameters

tl_camera_handle	The camera handle associated with the gain command.
gain	A value that is used to configure the gain setting.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.97 TL_CAMERA_SET_HOT_PIXEL_CORRECTION_THRESHOLD

```
typedef int(* TL_CAMERA_SET_HOT_PIXEL_CORRECTION_THRESHOLD) (void *tl_camera_handle, int hot_pixel_correction_threshold)
```

This function may be used to set the current threshold value for hot-pixel correction.

This value is a quantitative measure of how aggressively the camera will remove hot pixels.

Parameters

tl_camera_handle	The camera handle associated with the hot pixel correction threshold command.
hot_pixel_correction_threshold	A reference that specifies the current value of the hot pixel correction threshold for the specified camera.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.98 TL_CAMERA_SET_IMAGE_POLL_TIMEOUT

```
typedef int(* TL_CAMERA_SET_IMAGE_POLL_TIMEOUT) (void *tl_camera_handle, int timeout_ms)
```

Sets the current camera image poll time out value in milliseconds.

Parameters

tl_camera_handle	The camera handle associated with the exposure request.
timeout_ms	The time out value to be set.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.99 TL_CAMERA_SET_IS_EEP_ENABLED

```
typedef int(* TL_CAMERA_SET_IS_EEP_ENABLED) (void *tl_camera_handle, int is_eep_enabled)
```

Enables or disables the EEP operating mode.

Parameters

tl_camera_handle	The camera handle associated with the EEP request.
is_eep_enabled	A value that enables or disables EEP. 0 (zero) to disable EEP and 1 (one) to enable EEP.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.100 TL_CAMERA_SET_IS_FRAME_RATE_CONTROL_ENABLED

```
typedef int(* TL_CAMERA_SET_IS_FRAME_RATE_CONTROL_ENABLED) (void *tl_camera_handle, int is_enabled)
```

Enables or disables frame rate control.

Parameters

tl_camera_handle	The camera handle associated with the frame rate control request.
is_enabled	A value that enables or disables frame rate control. 0 (zero) to disable frame rate control and 1 (one) to enable frame rate control.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.101 TL_CAMERA_SET_IS_HOT_PIXEL_CORRECTION_ENABLED

```
typedef int(* TL_CAMERA_SET_IS_HOT_PIXEL_CORRECTION_ENABLED) (void *tl_camera_handle, int is_hot_pixel_correction_enabled)
```

This function may be used to set the current state of hot-pixel correction.

Parameters

tl_camera_handle	The camera handle associated with the hot pixel correction command.
is_hot_pixel_correction_enabled	A value that specifies the state of the hot pixel correction functionality for the specified camera. A 0 (zero) value disables hot pixel correction and a 1 (one) enables hot pixel correction.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.102 TL_CAMERA_SET_IS_LED_ON

```
typedef int(* TL_CAMERA_SET_IS_LED_ON) (void *tl_camera_handle, int is_led_on)
```

Some scientific cameras include an LED indicator light on the back panel.

This function sets the LED.

Parameters

tl_camera_handle	The camera handle associated with the LED request.
is_led_on	A value that controls the LED. 0 (zero) for off and 1 (one) for on.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.103 TL_CAMERA_SET_NAME

```
typedef int(* TL_CAMERA_SET_NAME) (void *tl_camera_handle, char *name)
```

Sets the camera name.

Parameters

tl_camera_handle	The camera handle associated with the name request.
name	A pointer to a character string containing the new camera name.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.104 TL_CAMERA_SET_NIR_BOOST_ENABLE

```
typedef int(* TL_CAMERA_SET_NIR_BOOST_ENABLE) (void *tl_camera_handle, int nir_boost_enable)
```

Enable or disable near-infrared-boost mode.

Parameters

tl_camera_handle	The camera handle associated with the cooling mode set request.
nir_boost_enable	A value that enables/disables NIR boost mode. 0 (zero) for off and 1 (one) for on.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.105 TL_CAMERA_SET_OPERATION_MODE

```
typedef int(* TL_CAMERA_SET_OPERATION_MODE) (void *tl_camera_handle, enum TL_CAMERA_OPERATION_MODE operation_mode)
```

Gets the operation mode of the camera.

0.3.1.1.106 TL_CAMERA_SET_ROI

```
typedef int(* TL_CAMERA_SET_ROI) (void *tl_camera_handle, int upper_left_x_pixels, int upper_left_y_pixels, int lower_right_x_pixels, int lower_↩
right_y_pixels)
```

Sets the current Region of Interest (ROI) values.

The ROI is specified by 2 sets of x,y coordinates that establish a bounded rectangular area:

- 1 for the upper left corner
- 1 for the lower right corner.

Parameters

tl_camera_handle	The camera handle associated with the ROI request.
upper_left_x_pixels	The x coordinate of the upper left corner of the ROI.
upper_left_y_pixels	The y coordinate of the upper left corner of the ROI.
lower_right_x_pixels	The x coordinate of the lower right corner of the ROI.
lower_right_y_pixels	The y coordinate of the lower right corner of the ROI.

Returns

0 if successful or a positive integer error code to indicate failure. In case of error, call tl_camera_get_last_error to get details. This error string is valid until another API called on the same thread.

0.3.1.1.107 TL_CAMERA_SET_TAP_BALANCE_ENABLE

```
typedef int(* TL_CAMERA_SET_TAP_BALANCE_ENABLE) (void *tl_camera_handle, int taps_balance_enable)
```

Sets the camera tap balance value.

Parameters

tl_camera_handle	The camera handle associated with the taps request.
taps_balance_enable	A value that enables/disables the tap balance feature. 0 (zero) for off and 1 (one) for on.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.108 TL_CAMERA_SET_TAPS

```
typedef int(* TL_CAMERA_SET_TAPS) (void *tl_camera_handle, enum TL_CAMERA_TAPS taps)
```

Sets the camera taps value.

Parameters

tl_camera_handle	The camera handle associated with the taps request.
taps	A TL_CAMERA_TAPS enumeration value.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.109 TL_CAMERA_SET_TRIGGER_POLARITY

```
typedef int(* TL_CAMERA_SET_TRIGGER_POLARITY) (void *tl_camera_handle, enum TL_CAMERA_TRIGGER_POLARITY trigger_polarity_enum)
```

Sets the current hardware trigger polarity for the specified camera.

Parameters

tl_camera_handle	The camera handle associated with the hardware trigger mode request.
trigger_polarity_enum	A TRIGGER_POLARITY value that is used to configure the trigger polarity.

Returns

0 if successful or a positive integer error code to indicate failure.

0.3.1.1.110 TL_CAMERA_SET_USER_MEMORY

```
typedef int(* TL_CAMERA_SET_USER_MEMORY) (void *tl_camera_handle, unsigned char *source_data_buffer, long long number_of_bytes_to_write, long long
camera_user_memory_offset_bytes)
```

Write to the on-camera, non-volatile memory that is available to the user.

Use tl_camera_get_user_memory_maximum_size to query the available memory.

Non-volatile memory can handle many writes, but the total number of writes is finite. Avoid unnecessary writes.

Parameters

tl_camera_handle	The camera handle associated with the USB port type request.
source_data_buffer	A byte buffer from which to write up to the specified number of bytes.
number_of_bytes_to_write	The number of bytes of the given data buffer to write at the given offset into on-camera, non-volatile, user-store memory.
camera_user_memory_offset_bytes	A byte offset in the on-camera, non-volatile memory from which to start reading. Use a value of zero to read from the beginning of user memory.

Returns

0 if successful or a positive integer error code to indicate failure.

See also

TL_CAMERA_GET_USER_MEMORY_MAXIMUM_SIZE, TL_CAMERA_GET_USER_MEMORY

0.3.1.2 Enumeration Type Documentation

0.3.1.2.1 TL_CAMERA_COMMUNICATION_INTERFACE

`enum TL_CAMERA_COMMUNICATION_INTERFACE`

The TL_CAMERA_COMMUNICATION_INTERFACE enumeration defines the values the SDK uses for specifying the physical camera interface.

Enumerator

TL_CAMERA_COMMUNICATION_INTERFACE_GIG_E	The camera uses the GigE Vision (GigE) interface standard.
TL_CAMERA_COMMUNICATION_INTERFACE_CAMERA_LINK	The camera uses the CameraLink serial-communication-protocol standard.
TL_CAMERA_COMMUNICATION_INTERFACE_USB	The camera uses a USB interface.
TL_CAMERA_COMMUNICATION_INTERFACE_MAX	Marks the end of the enumeration. Do not use.

0.3.1.2.2 TL_CAMERA_DATA_RATE

`enum TL_CAMERA_DATA_RATE`

The TL_CAMERA_DATA_RATE enumeration defines the options for setting the desired image data delivery rate.

Enumerator

TL_CAMERA_DATA_RATE_READOUT_FREQUENCY_20	Sets the device to an image readout frequency of 20 MHz.
TL_CAMERA_DATA_RATE_READOUT_FREQUENCY_40	Sets the device to an image readout frequency of 40 MHz.
TL_CAMERA_DATA_RATE_FPS_30	Sets the device to deliver images at 30 frames per second.
TL_CAMERA_DATA_RATE_FPS_50	Sets the device to deliver images at 50 frames per second.
TL_CAMERA_DATA_RATE_MAX	Marks the end of the enumeration. Do not use.

0.3.1.2.3 TL_CAMERA_EEP_STATUS

`enum TL_CAMERA_EEP_STATUS`

The TL_CAMERA_EEP_STATUS enumeration defines the options available for specifying the device's EEP mode. Equal Exposure Pulse (EEP) mode is an LVTTL-level signal that is active during the time when all rows have been reset during rolling reset, and the end of the exposure time (and the beginning of rolling readout). The signal can be used to control an external light source that will be on only during the equal exposure period, providing the same amount of exposure for all pixels in the ROI.

When EEP mode is disabled, the status will always be EEPStatus.Off.
EEP mode can be enabled, but, depending on the exposure value, active or inactive.
If EEP is enabled in bulb mode, it will always give a status of Bulb.

Enumerator

TL_CAMERA_EEP_STATUS_DISABLED	EEP mode is disabled.
TL_CAMERA_EEP_STATUS_ENABLED_ACTIVE	EEP mode is enabled and currently active.
TL_CAMERA_EEP_STATUS_ENABLED_INACTIVE	EEP mode is enabled, but due to an unsupported exposure value, currently inactive.
TL_CAMERA_EEP_STATUS_ENABLED_BULB	EEP mode is enabled in bulb mode.
TL_CAMERA_EEP_STATUS_MAX	Marks the end of the enumeration. Do not use.

0.3.1.2.4 TL_CAMERA_ERROR

`enum TL_CAMERA_ERROR`

The CAMERA_ERROR enumeration defines tl_camera_sdk error codes that can be returned from function calls.

Enumerator

TL_CAMERA_ERROR_NONE	The command request to the camera succeeded with no errors.
TL_CAMERA_ERROR_COMMAND_NOT_FOUND	The camera received an unknown command.
TL_CAMERA_ERROR_TOO_MANY_ARGUMENTS	The camera encountered too MANY arguments for the specified command.
TL_CAMERA_ERROR_NOT_ENOUGH_ARGUMENTS	The camera encountered too FEW arguments for the specified command.
TL_CAMERA_ERROR_INVALID_COMMAND	The camera received an invalid command.
TL_CAMERA_ERROR_DUPLICATE_COMMAND	The camera received a duplicate command.
TL_CAMERA_ERROR_MISSING_JSON_COMMAND	The camera received a command that is not documented in JSON.
TL_CAMERA_ERROR_INITIALIZING	The camera rejected the request because the it is being initialized.
TL_CAMERA_ERROR_NOTSUPPORTED	The user specified an unsupported and/or unknown command argument.

Enumerator

TL_CAMERA_ERROR_FPGA_NOT_PROGRAMMED	The camera rejected the request because the FPGA has not been programmed with a firmware image.
TL_CAMERA_ERROR_ROI_WIDTH_ERROR	The user specified an invalid ROI width.
TL_CAMERA_ERROR_ROI_RANGE_ERROR	The user specified an invalid ROI range.
TL_CAMERA_ERROR_RANGE_ERROR	The user specified an invalid range for the specified command.
TL_CAMERA_ERROR_COMMAND_LOCKED	The camera rejected the request because use of the specified command is restricted.
TL_CAMERA_ERROR_CAMERA_MUST_BE_STOPPED	The camera rejected the request because the specified command can only be accepted when the camera is stopped.
TL_CAMERA_ERROR_ROI_BIN_COMBO_ERROR	The camera encountered an ROI/binning error.
TL_CAMERA_ERROR_IMAGE_DATA_SYNC_ERROR	The camera encountered an image data sync error.
TL_CAMERA_ERROR_CAMERA_MUST_BE_DISARMED	The camera rejected the request because the specified command can only be accepted when the camera is disarmed.
TL_CAMERA_ERROR_MAX_ERRORS	Marks the end of the enumeration. Do not use.

0.3.1.2.5 TL_CAMERA_OPERATION_MODE

`enum TL_CAMERA_OPERATION_MODE`

The TL_CAMERA_OPERATION_MODE enumeration defines the available mode for camera. To determine which modes a camera supports, use tl_camera_get_is_operation↩ _mode_supported().

Enumerator

TL_CAMERA_OPERATION_MODE_SOFTWARE_TRIGGERED	Use software operation mode to generate one or more frames per trigger or to run continuous video mode.
TL_CAMERA_OPERATION_MODE_HARDWARE_TRIGGERED	Use hardware triggering to generate one or more frames per trigger by issuing hardware signals.
TL_CAMERA_OPERATION_MODE_BULB	Use bulb-mode triggering to generate one or more frames per trigger by issuing hardware signals. Please refer to the camera manual for signaling details.
TL_CAMERA_OPERATION_MODE_RESERVED1	Reserved for internal use.
TL_CAMERA_OPERATION_MODE_RESERVED2	Reserved for internal use.
TL_CAMERA_OPERATION_MODE_MAX	Marks the end of the enumeration. Do not use.

0.3.1.2.6 TL_CAMERA_SENSOR_TYPE

`enum TL_CAMERA_SENSOR_TYPE`

This describes the physical capabilities of the camera sensor.

Enumerator

TL_CAMERA_SENSOR_TYPE_MONOCHROME	Each pixel of the sensor indicates an intensity.
TL_CAMERA_SENSOR_TYPE_BAYER	The sensor has a bayer-patterned filter overlaying it, allowing the camera SDK to distinguish red, green, and blue values.
TL_CAMERA_SENSOR_TYPE_MONOCHROME_POLARIZED	The sensor has a polarization filter overlaying it allowing the camera to capture polarization information from the incoming light.
TL_CAMERA_SENSOR_TYPE_MAX	Marks the end of the enumeration. Do not use.

0.3.1.2.7 TL_CAMERA_TAPS

`enum TL_CAMERA_TAPS`

Scientific CCD cameras support one or more taps.

After exposure is complete, a CCD pixel array holds the charge corresponding to the amount of light collected at each pixel location. The data is then read out through 1, 2, or 4 channels at a time.

Enumerator

TL_CAMERA_TAPS_SINGLE_TAP	Charges are read out through a single analog-to-digital converter.
TL_CAMERA_TAPS_DUAL_TAP	Charges are read out through two analog-to-digital converters.
TL_CAMERA_TAPS_QUAD_TAP	Charges are read out through four analog-to-digital converters.
TL_CAMERA_TAPS_MAX_TAP	Marks the end of the enumeration. Do not use.

0.3.1.2.8 TL_CAMERA_TRIGGER_POLARITY

enum TL_CAMERA_TRIGGER_POLARITY

The TRIGGER_POLARITY enumeration defines the options available for specifying the hardware trigger polarity.

These values specify which edge of the input trigger pulse that will initiate image acquisition.

Enumerator

TL_CAMERA_TRIGGER_POLARITY_ACTIVE_HIGH	Acquire an image on the RISING edge of the trigger pulse.
TL_CAMERA_TRIGGER_POLARITY_ACTIVE_LOW	Acquire an image on the FALLING edge of the trigger pulse.
TL_CAMERA_TRIGGER_POLARITY_MAX	Marks the end of the enumeration. Do not use.

0.3.1.2.9 TL_CAMERA_USB_PORT_TYPE

enum TL_CAMERA_USB_PORT_TYPE

The TL_CAMERA_USB_PORT_TYPE enumeration defines the values the SDK uses for specifying the USB bus speed.

These values are returned by SDK API functions and callbacks based on the type of physical USB port that the device is connected to.

Enumerator

TL_CAMERA_USB_PORT_TYPE_USB1_0	The device is connected to a USB 1.0/1.1 port (1.5 Mbits/sec or 12 Mbits/sec).
TL_CAMERA_USB_PORT_TYPE_USB2_0	The device is connected to a USB 2.0 port (480 Mbits/sec).
TL_CAMERA_USB_PORT_TYPE_USB3_0	The device is connected to a USB 3.0 port (5000 Mbits/sec).
TL_CAMERA_USB_PORT_TYPE_MAX	Marks the end of the enumeration. Do not use.

Index