THORLABS

Scientific Imaging

Application Programming Interface (API)

Table of Contents

Chapter 1 Description

1.1. TSI Camera SDK

The TSI Imaging SDK consists of Libraries, Drivers, Documentation, and Examples that make it easy to incorporate a TSI camera into your system.

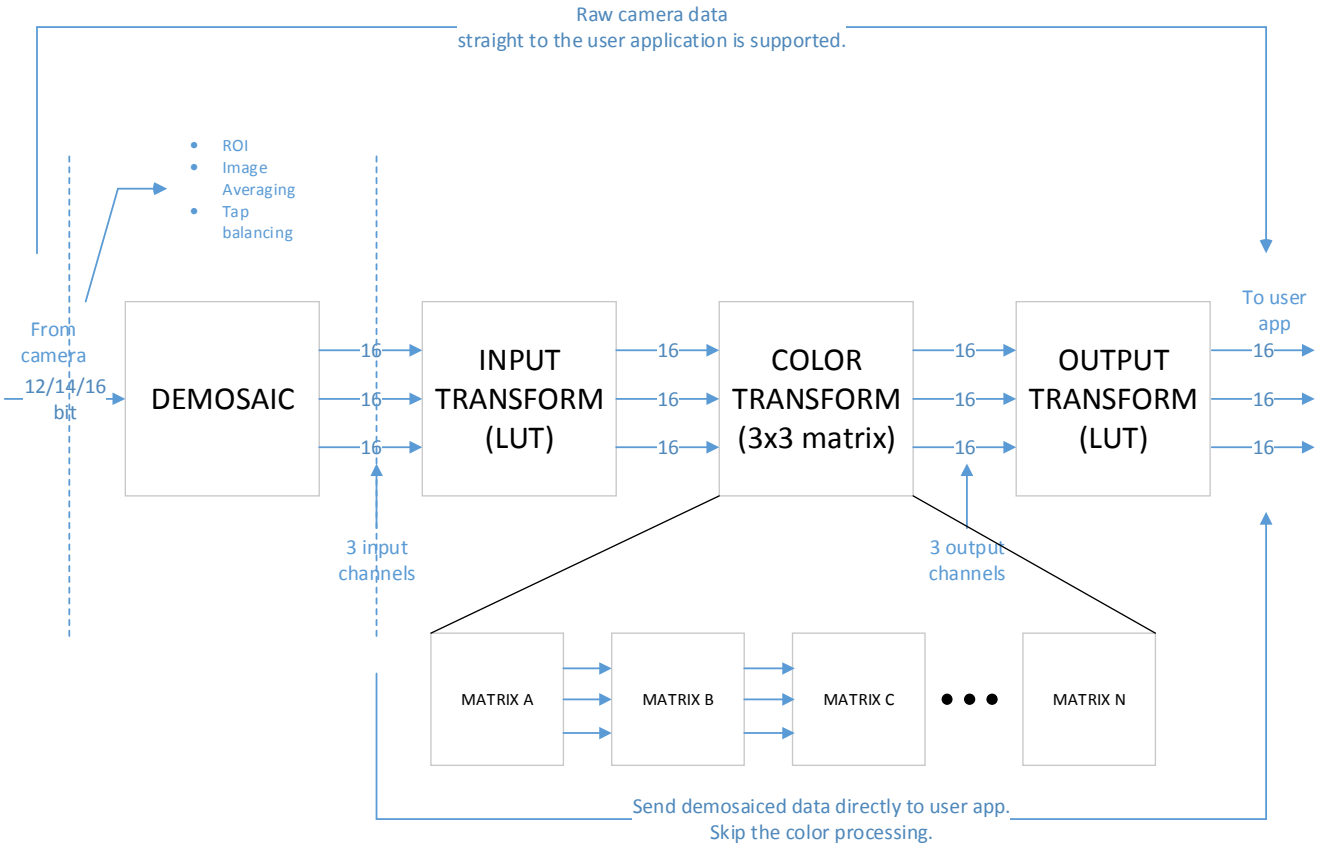
1.1.1. Imaging Concepts

Basic steps to get image from camera:

1) TsiSDK *sdk = get_tsi_sdk(0); //Obtain a reference to the sdk using the static function
2) bool success = sdk->Open(); //Open the SDK
3) int numCameras = sdk->GetNumberOfCameras(); //Get the number of available cameras
4) TsiCamera *cam = sdk->GetCamera(0..numCameras-1); //Get a reference to a camera
    a. cam->Open(); //Open the camera
    b. Register for desired events (ImageNotification, ControlCallback, etc)
    c. Set/Get desired camera parameters
    d. cam->Start();
    e. Acquire images (see step 5)
    f. cam->Stop();
5) Access the TsiImage from the appropriate callback and use the pointer to the data. The callback function must complete its processing fast enough to be ready to process the next incoming frame – if this timing constraint is not met, then the SDK will not function correctly. If lengthy image processing is required, it is highly recommended that the host application copy the obtained image from inside the callback function, release the image and then exit. The copy of the image data can be processed elsewhere (e.g. on another thread).
    a. TsiImage *image = cam->GetPendingImage(); //Call GetPendingImage in Notify Callback
        i. Alternatively, just receive the image in the parameters of Image Callback
        ii. Make sure image pointer isn't null
        iii. Check error codes in callbacks as necessary
    b. Remember to call cam->FreeImage(image); ASAP to clear the buffer for incoming images
6) cam->Close(); //Close the camera when you are done
7) sdk->Close(); //Close the SDK when done
8) release_tsi_sdk(sdk); //Release the resources used by the SDK

Color Pipeline Operation

The following diagram illustrates the steps that the SDK uses to process a color image:



1) Demosaic:

The SDK will demosaic the image from the camera using a standard algorithm.

2) Input transform:

The host application can supply an input transform function in the form of a 1 dimentional LUT that the SDK will apply to every channel of the demosaiced color pixel data. The size of the LUT is constrained to the native camera pixel data bit depth.

An example use of this step would be to linearize the demosaiced data prior to the color transform step.

3) Color transform:

The host application can specify 1 or more 3x3 matrices that the SDK can use to process the incoming desmosaiced data. If multiple matrices are specified, the SDK concatenates them into a single matrix which is then applied to the incoming frame data.

4) Output transform:

The host application can supply an output transform function in the form of a 1 dimensional LUT that the SDK will apply to the demosaiced color pixel data. The size of the LUT is constrained to the native camera pixel data bit depth.

An example application of this step would be to nonlinearize the output data via the application of a gamma function.

The pipeline operation is configurable in the following ways:

1) If the host application does not want the SDK to perform any color processing, then it can simply obtain the raw images using a TsiCamera object similar to how images are obtained for monochrome images.

2) The host can request the SDK to only demosaic the data and skip the other color processing steps.

3) The host can leverage the full capabilities of the SDK by configuring the color pipeline to demosaic and process the image.

Important note: The SDK does not support binning selections other than 1x1 for color cameras with a single chip sensor.

Basic steps to get a color image from a color enabled camera:

1) TsiSDK *sdk = get_tsi_sdk(0); //Obtain a reference to the sdk using the static function

2) bool success = sdk->Open(); //Open the SDK

3) int numCameras = sdk->GetNumberOfCameras(); //Get the number of available cameras

4) TsiCamera *cam = sdk->GetCamera(0..numCameras-1); //Get a reference to a camera

    a. cam->Open(); //Open the camera

    b. cam->GetParameter (TSI_COLOR_CAMERA_ATTRIBUTES, sizeof(ColorCameraAttributes), &c); // Ask the camera if it is color enabled. The variable c is an instance of type ColorCameraAttributes.

    c. bool isColorCamera = strcmp (c.ColorFilterArrayType, “mono”) != 0;

    d. TsiColorCamera* colorCam = (TsiColorCamera*) cam; // convert to a TsiColorCamera if isColorCamera is true. If isColorCamera is false, then the camera is not color enabled and only monochrome images can be obtained – please refer to the previous procedure for obtaining monochrome images.

    e. Register for desired events (ImageNotification, ControlCallback, etc)

    f. Set/Get desired camera parameters

    g. Configure the color pipeline. The steps below are for illustrative purposes only; what follows is a subset of what the SDK offers in terms of color pipeline configurability.

        i. colorCam->ConcatenateColorTransform (Tsi_Camera_Color_Correction, 0); // Add the camera specific color correction matrix.

        ii. colorCam->ConcatenateColorTransform (TSI_sRGB, 0); // Add the color matrix and nonlinearization step for the sRGB output color space. The 0 indicates that the output bit depth should be set to the default value for the current camera configuration..

h. colorCam->Start();

i. Acquire images (see step 5)

j. colorCam->Stop();

5) Access the TsiColorImage from the appropriate callback and use the pointer to the data.

   a. TsiColorImage *colorImage = colorCam->GetPendingColorImage(); //Call GetPendingColorImage in Notify Callback

      i. Make sure image pointer isn't null

      ii. Check error codes in callbacks as necessary

   b. Remember to call colorCam->FreeColorImage(colorImage); ASAP to clear the buffer for incoming images

6) colorCam->Close(); //Close the camera when you are done

7) sdk->Close(); //Close the SDK when done

8) release_tsi_sdk(sdk); //Release the resources used by the SDK

Chapter 2 C++ API

2.1. Enumerations

2.1.1. TS_ATTR_ID

```
typedef enum _TSI_ATTR_ID
{
    TSI_ATTR_NAME,
    TSI_ATTR_DATA_TYPE,
    TSI_ATTR_ARRAY_COUNT,
    TSI_ATTR_FLAGS,
    TSI_ATTR_MIN_VALUE,
    TSI_ATTR_MAX_VALUE,
    TSI_ATTR_DEFAULT_VALUE,

    TSI_MAX_ATTR

} TSI_ATTR_ID, *PTSI_ATTR_ID;
```

2.1.2. TSI_DATA_TYPE

```
typedef enum _TSI_DATA_TYPE
{
    TSI_TYPE_NONE,
    TSI_TYPE_UNS8,
    TSI_TYPE_UNS16,
    TSI_TYPE_UNS32,
    TSI_TYPE_UNS64,
    TSI_TYPE_INT8,
    TSI_TYPE_INT16,
    TSI_TYPE_INT32,
    TSI_TYPE_INT64,
    TSI_TYPE_TEXT,
    TSI_TYPE_FP,

    TSI_MAX_TYPES

} TSI_DATA_TYPE;
```

2.1.3. TSI_PARAM_FLAGS

```
typedef enum _TSI_PARAM_FLAGS
{
    TSI_FLAG_READ_ONLY          = 0x00000001,
    TSI_FLAG_WRITE_ONLY         = 0x00000002,
    TSI_FLAG_UNSUPPORTED        = 0x00000004,
    TSI_FLAG_VALUE_CHANGED      = 0x00000008

} TSI_PARAM_FLAGS;
```

2.1.4. TSI_CAMERA_STATUS

```
typedef enum _TSI_CAMERA_STATUS
{
    TSI_STATUS_CLOSED,
    TSI_STATUS_OPEN,
    TSI_STATUS_BUSY,

    TSI_STATUS_MAX

} TSI_CAMERA_STATUS;
```

2.1.5. TSI_ERROR_CODE

```
typedef enum _TSI_ERROR_CODE
{
    TSI_NO_ERROR,
    TSI_ERROR_UNKNOWN,

    TSI_ERROR_UNSUPPORTED,

    TSI_ERROR_PARAMETER_UNSUPPORTED,
    TSI_ERROR_ATTRIBUTE_UNSUPPORTED,

    TSI_ERROR_INVALID_ROI,
    TSI_ERROR_INVALID_BINNING,

    TSI_ERROR_INVALID_PARAMETER_UNDERFLOW,
    TSI_ERROR INVALID_PARAMETER_OVERFLOW,

    TSI_ERROR_CAMERA_COMM_FAILURE,

    TSI_ERROR_CAMERA_INVALID_DATA,

    TSI_ERROR_NULL_POINTER_SUPPLIED,
    TSI_CAMERA_INVALID_DATA_SIZE_OR_TYPE,
    TSI_ERROR_IMAGE_BUFFER_OVERFLOW,

    TSI_ERROR_INVALID_NUMBER_OF_IMAGE_BUFFERS,
    TSI_ERROR_IMAGE_BUFFER_ALLOCATION_FAILURE
    TSI_ERROR_TOO_MANY_IMAGE_BUFFERS,
    TSI_MAX_ERROR

} TSI_ERROR_CODE, *PTSI_ERROR_CODE;
```

2.1.6. TSI_ACQ_STATUS_ID

```
typedef enum _TSI_ACQ_STATUS_ID
{
    TSI_ACQ_STATUS_IDLE,
    TSI_ACQ_STATUS_WAITING_FOR_TRIGGER,
    TSI_ACQ_STATUS_EXPOSING,
    TSI_ACQ_STATUS_READING_OUT,
    TSI_ACQ_STATUS_DONE,
    TSI_ACQ_STATUS_ERROR,
    TSI_ACQ_STATUS_TIMEOUT,
    TSI_MAX_ACQ_STATUS_ID

} TSI_ACQ_STATUS_ID;
```

2.1.7. TSI_CAMERA_CONTROL_EVENT_ID

```
typedef enum _TSI_CAMERA_CONTROL_EVENT_ID
{
    TSI_CAMERA_CONTROL_EXPOSURE_START    ,
    TSI_CAMERA_CONTROL_EXPOSURE_COMPLETE ,
    TSI_CAMERA_CONTROL_SEQUENCE_START    ,
    TSI_CAMERA_CONTROL_SEQUENCE_COMPLETE ,
    TSI_CAMERA_CONTROL_READOUT_START     ,
    TSI_CAMERA_CONTROL_READOUT_COMPLETE  ,

    TSI_MAX_CAMERA_CONTROL_EVENT_ID

} TSI_CAMERA_CONTROL_EVENT_ID;
```

2.1.8. TSI_IMAGE_NOTIFICATION_EVENT_ID

```
typedef enum _TSI_IMAGE_NOTIFICATION_EVENT_ID
{
    TSI_IMAGE_NOTIFICATION_PENDING_IMAGE,
    TSI_IMAGE_NOTIFICATION_ACQUISITION_ERROR,
    TSI_MAX_IMAGE_NOTIFICATION_EVENT_ID

} TSI_IMAGE_NOTIFICATION_EVENT_ID;
```

2.1.9. TSI_PARAM_ID

```
typedef enum _TSI_PARAM_ID
{
    TSI_PARAM_CMD_ID_ATTR_ID       ,
    TSI_PARAM_ATTR                 ,
    TSI_PARAM_PROTOCOL             ,
    TSI_PARAM_FW_VER               ,
```

```
    TSI_PARAM_HW_VER               ,
    TSI_PARAM_HW_MODEL             ,
    TSI_PARAM_HW_SER_NUM           ,
    TSI_PARAM_CAMSTATE             ,
    TSI_PARAM_CAM_EXPOSURE_STATE   ,
    TSI_PARAM_CAM_TRIGGER_STATE    ,
    TSI_PARAM_EXPOSURE_UNIT        ,
    TSI_PARAM_EXPOSURE_TIME        ,
    TSI_PARAM_ACTUAL_EXPOSURE_TIME ,
    TSI_PARAM_HSIZE                ,
    TSI_PARAM_VSIZE                ,
    TSI_PARAM_ROI_BIN              ,
    TSI_PARAM_FRAME_COUNT          ,
    TSI_PARAM_CURRENT_FRAME        ,
    TSI_PARAM_OP_MODE              ,
    TSI_PARAM_CDS_GAIN_INDEX       ,
    TSI_PARAM_CDS_GAIN = TSI_PARAM_CDS_GAIN_INDEX,
    TSI_PARAM_VGA_GAIN             ,
    TSI_PARAM_GAIN                 ,
    TSI_PARAM_OPTICAL_BLACK_LEVEL  ,
    TSI_PARAM_PIXEL_OFFSET         ,
    TSI_PARAM_READOUT_SPEED_INDEX  ,
    TSI_PARAM_READOUT_SPEED        ,
    TSI_PARAM_FRAME_TIME           ,
    TSI_PARAM_FRAME_RATE           ,
    TSI_PARAM_COOLING_MODE         ,
    TSI_PARAM_COOLING_SETPOINT     ,
    TSI_PARAM_TEMPERATURE          ,
    TSI_PARAM_QX_OPTION_MODE       ,
    TSI_PARAM_TURBO_MODE             ,
    TSI_PARAM_TURBO_CODE_MODE = TSI_PARAM_TURBO_MODE,
    TSI_PARAM_XORIGIN              ,
    TSI_PARAM_YORIGIN              ,
    TSI_PARAM_XPIXELS              ,
    TSI_PARAM_YPIXELS              ,
    TSI_PARAM_XBIN                 ,
    TSI_PARAM_YBIN                 ,
    TSI_PARAM_IMAGE_ACQUISTION_MODE,
    TSI_PARAM_NAMED_VALUE          ,
    TSI_PARAM_TAPS_INDEX           ,
    TSI_PARAM_TAPS_VALUE           ,
    TSI_PARAM_RESERVED_1           ,
    TSI_PARAM_RESERVED_2           ,
    TSI_PARAM_RESERVED_3           ,
    TSI_PARAM_RESERVED_4           ,
    TSI_PARAM_GLOBAL_CAMERA_NAME   ,
    TSI_PARAM_CDS_GAIN_VALUE       ,
    TSI_PARAM_PIXEL_SIZE           ,
    TSI_PARAM_BITS_PER_PIXEL       ,
    TSI_PARAM_BYTES_PER_PIXEL      ,
    TSI_PARAM_READOUT_TIME         ,
    TSI_PARAM_HW_TRIGGER_ACTIVE    ,
    TSI_PARAM_HW_TRIG_SOURCE       ,
    TSI_PARAM_HW_TRIG_POLARITY     ,
   TSI_PARAM_TAP_BALANCE_ENABLE   ,
   TSI_PARAM_DROPPED_FRAMES       ,
   TSI_PARAM_EXPOSURE_TIME_US     ,
```

```
   TSI_PARAM_RESERVED_5           ,
   TSI_PARAM_RESERVED_6           ,
   TSI_PARAM_RESERVED_7           ,
   TSI_PARAM_UPDATE_PARAMETERS    ,
   TSI_PARAM_FEATURE_LIST         ,
   TSI_PARAM_FEATURE_VALID        ,
   TSI_PARAM_NUM_IMAGE_BUFFERS    ,
   TSI_PARAM_COLOR_FILTER_TYPE    ,
   TSI_PARAM_COLOR_FILTER_PHASE   ,
   TSI_PARAM_COLOR_IR_FILTER_TYPE ,
   TSI_PARAM_COLOR_CAMERA_CORRECTION_MATRIX ,
   TSI_PARAM_CCM_OUTPUT_COLOR_SPACE ,
   TSI_PARAM_DEFAULT_WHITE_BALANCE_MATRIX ,
   TSI_PARAM_USB_ENABLE_LED ,
   TSI_MAX_PARAMS

} TSI_PARAM_ID;
```

2.1.10. TSI_IMAGE_MODE

```
typedef enum _TSI_IMAGE_MODES
{
    TSI_IMAGE_MODE_ALLOCATE ,
    TSI_IMAGE_MODE_STREAM   ,
    TSI_IMAGE_MODE_TRIGGER  ,

    TSI_MAX_IMAGE_MODES

} TSI_IMAGE_ACQUISTION_MODES;
```

2.1.11. TSI_HW_TRIG_SOURCE

```
typedef enum _TSI_HW_TRIG_SOURCE
{
   TSI_HW_TRIG_OFF,
   TSI_HW_TRIG_AUX,
   TSI_HW_TRIG_CL,
   TSI_HW_TRIG_MAX
} TSI_HW_TRIG_SOURCE, *PTSI_HW_TRIG_SOURCE;
```

2.1.12. TSI_HW_TRIG_POLARITY

```
typedef enum _TSI_HW_TRIG_POLARITY
{
   TSI_HW_TRIG_ACTIVE_HIGH,
   TSI_HW_TRIG_ACTIVE_LOW,
   TSI_HW_TRIG_POL_MAX
} TSI_HW_TRIG_POLARITY, *PTSI_HW_TRIG_POLARITY;
```

2.1.13. TSI_OP_MODE

```
typedef enum _TSI_OP_MODE
{
   TSI_OP_MODE_NORMAL,
  TSI_OP_MODE_PDX,
  TSI_OP_MODE_TOE,
   TSI_OP_MODE_RESERVED_1,

} TSI_OP_MODE, *PTSI_OP_MODE;
```

2.1.14. TSI_EXPOSURE_UNITS

```
typedef enum _TSI_EXPOSURE_UNITS
{
   TSI_EXP_UNIT_MICROSECONDS,
   TSI_EXP_UNIT_MILLISECONDS,
   TSI_EXP_UNIT_MAX

} TSI_EXPOSURE_UNITS, *PTSI_EXPOSURE_UNITS;
```

2.1.15. TSI_ADDRESS_SELECT

```
typedef enum  _TSI_ADDRESS_SELECT
{
    TSI_ADDRESS_SELECT_IP,
    TSI_ADDRESS_SELECT_MAC,
    TSI_ADDRESS_SELECT_ADAPTER_ID,
    TSI_ADDRESS_SELECT_MAX

} TSI_ADDRESS_SELECT;
```

2.1.16. TSI_COLOR_FILTER_ARRAY_PHASE_VALUES

```
enum TSI_COLOR_FILTER_ARRAY_PHASE_VALUES
{
    CFA_PHASE_NOT_SUPPORTED,
    BAYER_RED,
    BAYER_BLUE,
    BAYER_GREEN_LEFT_OF_RED,
    BAYER_GREEN_LEFT_OF_BLUE
}
```

2.1.17. TSI_TRANSFORM

```
enum TSI_TRANSFORM
{
    TSI_sRGB,
    TSI_RGB_Linear,
    TSI_sRGB8_24,
    TSI_sRGB8_32,
    TSI_PCS_48,
    TSI_Mono_12,
    TSI_Mono_14,
    TSI_Mono_16,
    TSI_Camera_Color_Correction,
    TSI_Default_White_Balance,
    TSI_Quick_Color_Configuration
}
```

2.1.18. TSI_COLOR_PROCESSING_MODE

```
enum TSI_COLOR_PROCESSING_MODE
{
    TSI_COLOR_DEMOSAIC,
    TSI_COLOR_POST_PROCESS
}
```

2.2. Data Types

2.2.1. TSI_ROI_BIN

```
typedef struct _TSI_ROI_BIN {

    UNS32 XOrigin; // X Origin in sensor array
    UNS32 YOrigin; // Y Origin in sensor array
    UNS32 XPixels; // Frame width in pixels
    UNS32 YPixels; // Frame height in pixels

    UNS32 XBin;    // Binning factor in the X dimension
    UNS32 YBin;    // Binning factor in the Y dimension

} TSI_ROI_BIN, *PTSI_ROI_BIN;
```

2.2.2. TSI_FUNCTION_CAMERA_CONTROL_INFO

```
typedef struct _TSI_FUNCTION_CAMERA_CONTROL_INFO {

   uint32_t FrameNumber;

   struct {

     uint32_t Year;
     uint32_t Month;
     uint32_t Day;
     uint32_t Hour;
     uint32_t Min;
     uint32_t Sec;
     uint32_t MS;
     uint32_t US;

   } TimeStamp;

} TSI_FUNCTION_CAMERA_CONTROL_INFO,
*PTSI_FUNCTION_CAMERA_CONTROL_INFO;
```

2.2.3. TSI_FUNCTION_CAMERA_CONTROL_CALLBACK

```
typedef void (*TSI_FUNCTION_CAMERA_CONTROL_CALLBACK)
    (int  ctl_event,      void *context);
```

2.2.4. TSI_FUNCTION_CAMERA_CONTROL_CALLBACK_EX

```
typedef void (*TSI_FUNCTION_CAMERA_CONTROL_CALLBACK_EX )
(int  ctl_event,     TSI_FUNCTION_CAMERA_CONTROL_INFO
*ctl_event_info, void *context);
```

2.2.5. TSI_FUNCTION_IMAGE_NOTIFICATION_CALLBACK

```
typedef void (*TSI_FUNCTION_IMAGE_NOTIFICATION_CALLBACK)
   (int  notification,   void *context);
```

2.2.6. TSI_FUNCTION_IMAGE_CALLBACK

```
typedef void (*TSI_FUNCTION_IMAGE_CALLBACK)
    (TsiImage *tsi_image, void *context);
```

2.2.7. TSI_SET_CAMERA_CONNECT_CALLBACK

```
typedef int (*TSI_SET_CAMERA_CONNECT_CALLBACK)(void (*cb)(int
event, int cam_index, char* name, char* serial_num, char*
bus_str, void*), void *context);
```

2.3. TsiSdk Class

This class provides a framework for all the other classes in the API to exist within.

Instantiating an object of this type and then calling the OpenSDK method instructs the API to examine your system, searching for and building a list of cameras that it finds.

The CloseSDK (coupled with a subsequent call to OpenSDK) can be used to handle instances where the list of cameras changes dynamically.

The destructor for the TsiSdk class will first close and delete any camera objects that were allocated, and then close and clean up the SDK.

2.3.1. Constructor / Destructor

The constructor for the TsiSDK class initializes the library but does not actually build a list of cameras – this functionality is implemented in the OpenSDK method so that the TsiSdk class does not have to be deleted and recreated when the list of cameras changes.

The destructor for this class contains an implicit call to the CloseSDK (see below) method.

2.3.2. Methods

TsiSdk::Open

```
bool Open();
```

This method must be called before any other

TsiSdk::Close

```
bool Close();
```

This method iterates over the list of cameras, first closing any cameras that have been left open, then deleting their objects. After calling this method (or destructing an object of type TsiSdk), make sure your program does not access any TsiCamera objects.

TsiSdk::GetNumberOfCameras

```
int GetNumberOfCameras ();
```

This method is used after the OpenSDK method to get a count of the number of cameras the OpenSDK method found.

TsiSdk::GetCamera

```
TsiCamera* GetCamera (int CameraNumber);
```

This method fails if the value returned is NULL.

Otherwise, the pointer returned is to a camera object of type TsiCamera. See TsiCamera class definition below for more details.

TsiSdk::GetCameraInterfaceTypeStr

```
char* GetCameraInterfaceTypeStr (int camera_number );
```

This method returns a string describing the interface type of the camera.

TsiSdk::GetCameraAddressStr

```
char* GetCameraAddressStr (int camera_number,
                           TSI_ADDRESS_SELECT address_select );
```

This method returns the specified address type for a specified camera.

TsiSdk::GetCameraName

```
char* GetCameraName (int camera_number );
```

This method returns a string with the name of the selected camera.

TsiSdk::ElapsedTime

```
uint64 ElapsedTime (uint64 StartTime);
```

This method provides a convenient way to calculate the elapsed time in milliseconds between successive calls to this method.

Calling this method with a start time of zero establishes the start time. Calling the method again, using the results of the first call will result in the elapsed time in millisconds between those calls being returned.

TsiSdk::GetLastErrorStr

```
char* GetLastErrorStr();
```

Returns the string describing the last error.

TsiSdk::GetErrorCode

```
TSI_ERROR_CODE GetErrorCode();
```

Returns the current error code for the SDK, a returned value of zero indicates success. Clears any non-zero error code.

TsiSdk::ClearError

```
bool ClearError ();
```

Clears any existing error code for the SDK without having to read it using GetErrorCode.

TsiSdk::GetErrorString

```
bool GetErrorString(TSI_ERROR_CODE ErrorCode, char
*StringBuffer, int &StringLength)
```

Returns the non-localized ASCII string description of the supplied error code.

Passing a NULL pointer and pointer to a variable containing a length of zero length results in the length of the string associated with the supplied error code being returned so that a buffer of appropriate length can be dynamically allocated.

TsiSdk::GetUtilityObject

```
TsiUtil* GetUtilityObject ();
```

Returns a pointer to an instance of TsiUtil which has some functions for saving out 16-bit tif and png image formats.

2.4. TsiCamera Class

Objects of this class are used to control and acquire data from the cameras in your system. You cannot simply instantiate an object of this type. Instead, instantiate an object of the **TsiSdk** class, call the **OpenSDK** method, and then use the **GetCamera** method in that class to get a pointer to an object of this type.

2.4.1. METHODS

TsiCamera::Open

```
bool Open ();
```

Opens this camera for subsequent communications. The reason for opening and closing cameras is to conserve resources for cameras that have been enumerated, but are not currently in use.

This method must be called (once) before any other methods other than GetCameraName can be used.

TsiCamera::Close

```
bool Close ();
```

This method is called implicitly by the TsiSdk method CloseSDK and it's destructor, but one can call this method directly to close the camera to conserve resources when a camera is not needed.

TsiCamera::Status

```
bool Status (TSI_CAMERA_STATUS *CameraStatus);
```

This method returns the current status of the camera (see the definition of TSI_CAMERA_STATUS above). The intent of this method is to return the overall state of the camera (open, closed, busy, etc.).

For the status of an acquisition, use the GetStatus method of the TsiAcquistion object below.

TsiCamera::GetCameraName

```
char* GetCameraName ();
```

This function returns a pointer to the camera name. If no name has been previously assigned to the camera, a generic camera name will be returned.

TsiCamera::SetCameraName

```
bool SetCameraName (char *CameraName);
```

This function allows the caller to give the camera a user generated name. This function is intended for use in multi-camera systems where the user may want to assign a name to a particular camera that may have a particular function or be connected to a particular part of their system. This name is persistant and is tied to the model and serial number of the camera.

TsiCamera::GetParameter

```
bool GetParameter (TSI_PARAM_ID ParameterID, size_t Length, void
*data);
```

This method is the main way of querying the various camera parameters, discovering whether a particular setting is supported on a camera, and other attributes about the parameter, such as it's data type, length (in the case of vector or array type), and any other flags the parameter might have.

To use this method, you must supply a parameter identifier (see TSI_PARAMETER_ID as defined above), the size of the variable to hold the data, and a pointer to the variable to hold the data returned.

Attributes ***about*** the parameter, with the exception of the minimum and maximum values a parameter may have all have fixed sizes. The actual value, the min and the max values will all have the same type, and that type can be discovered by using the **TSI_ATTR_DATA_TYPE** attribute identifier. The value retuned will be a **TSI_DATA_TYPE** enumerated type, The GetDataTypeSize method can be used to determine the scalar size for the type. The total storage required for a parameter will be the size for the data type multiplied by the value returned by **TSI_ATTR_DATA_COUNT** attribute id.

TsiCamera::GetDataTypeSize

```
int32 GetDataTypeSize (TSI_DATA_TYPE DataType);
```

This method returns the size in bytes of any of the SDK defined data types.

TsiCamera::SetTextCommand

```
bool SetTextCommand (char *str);
```

This method sends a serial text command directly to the camera.

TsiCamera::SetTextCallback

```
bool SetTextCommand (TSI_TEXT_CALLBACK_FUNCTION func, void
*context);
```

Using this method, you can register for a callback to receive serial text responses to serial text commands issued via SetTextCommand(char *str).

TsiCamera::SetParameter

```
bool SetParameter (TSI_PARAM_ID ParameterID, void *data);
```

This method is a sister function to the GetParameter function described above and is used to set any parameter that is does not have the **TSI_FLAG_READ_ONLY** flag set.

A parameter's setting may be cached (unless the API determines it's value is volatile and must be sent to the camera each time, even if the value does not actually change). This is used to cut down on unnecessary traffic to / from the camera.

Normally, any changes to the following parameters are not applied until the camera is started via a call to TsiCamera.Start:

- TSI_PARAM_READOUT_SPEED
- TSI_PARAM_TAPS_INDEX
- TSI_PARAM_ROI_BIN
- TSI_PARAM_TURBO_CODE_MODE

If it is necessary to force the camera to apply changes to these parameters without (or prior to) starting it, that can be accomplished by setting the TSI_PARAM_UPDATE_PARAMETERS parameter.

TsiCamera::SetCameraControlCallback

```
bool SetCameraControlCallback (
   TSI_FUNCTION_CAMERA_CONTROL_CALLBACK   func,
   void *                                 context
   );
```

This method allows the caller to specify a callback function for interesting events about the camera.

See the **TSI_CAMERA_CONTROL_EVENT_ID** enumeration above for the supported list of events.

The **TSI_FUNCTION_CAMERA_CONTROL_CALLBACK** type definition above describes the function header for the callback function.

TsiCamera::SetCameraControlCallbackEx

```
bool SetCameraControlCallbackEx (
   TSI_FUNCTION_CAMERA_CONTROL_CALLBACK_EX   func,
```

```
    void *                                  context
    );
```

This method allows the caller to specify a callback function for interesting events about the camera.

See the **TSI_CAMERA_CONTROL_EVENT_ID** enumeration above for the supported list of events.

The **TSI_FUNCTION_CAMERA_CONTROL_CALLBACK_EX** type definition above describes the function header for the callback function.

TsiCamera::SetImageNotificationCallback

```
bool SetImageNotificationCallback (
    TSI_FUNCTION_IMAGE_NOTIFICATION_CALLBACK func,
    Void *                                   context
    );
```

Using this method, you can register callbacks for events such as cameras being unplugged or other asynchronous events occurring on the camera.

See the **TSI_IMAGE_NOTIFICATION_EVENT_ID** enumeration above for the supported list of events.

The **TSI_FUNCTION_IMAGE_NOTIFICATION_CALLBACK** type definition above describes the function header for the callback function.

The supplied notification callback function must execute fast. The mechanism was not designed to handle high latency callback functions and if such a function is supplied, this could result in anomalous behavior that is considered undefined and therefore not supported.

TsiCamera::SetImageCallback

```
bool SetImageCallback (
    TSI_FUNCTION_IMAGE_CALLBACK             func,
    Void *                                   context
    );
```

This method allows the caller to specify a callback function that will be called whenever there is a new frame of image data available.

The **TSI_FUNCTION_IMAGE_CALLBACK** type definition above describes the function header for the callback function.

The supplied image callback function must execute fast. The mechanism was not designed to handle high latency callback functions and if such a function is supplied, this could result in anomalous behavior that is considered undefined and therefore not supported.

TsiCamera::ResetCamera

```
bool ResetCamera ();
```

This method is used to set the camera to it's power-on state. The application will have to query the camera parameters to determine what the current camera settings are.

This can be accomplished by iterating over all the camera settings calling a GetParameter with the Parameter IDs the application cares about, specifying the TSI_ATTR_FLAGS attribute, and checking for the ATTR_VALUE_CHANGED flag. Any parameter that has changed as a result of restoring the camera to it's power on state will have this flag set.

TsiCamera::FreeImage

```
bool FreeImage (TsiImage *Image);
```

This method frees the image data object. The Close method and the destructor for this class will close and free up any image data objects that have been created. Any pointers to TsiImage objects that have been created must not be accessed after the object has been freed by the TsiCamera FreeImageData or Close methods or the TsiSdk object's CloseSDK method or destructor.

TsiCamera::FreeAllPendingImages

```
bool FreeAllPendingImages (TsiImage *Image);
```

This method frees all of the the pending images in the buffer.

TsiCamera::GetPendingImage

```
TsiImage* GetPendingImage (void);
```

This method will return a pointer to the next available image.

TsiCamera::GetLastPendingImage

```
TsiImage* GetLastPendingImage (void);
```

This method will return a pointer to the last available image.

TsiCamera::StartAndWait

```
bool StartAndWait (int32 Timeout);
```

This method starts an aquistition of one or more frames using the current camera parameters and waits for it to finish. If the elapsed time to acquire the image data exceeds the timeout value (in milliseconds), the acquisition will be cancelled and the function will return FALSE.

Also, if the combination of current camera parameters do not permit proper operation of the camera, FALSE will be returned. An example would be calling StartAcquistionAndWait with the TSI_PARAM_FRAME_COUNT parameter set to 0 (which indicates acquire until a StopCamera call is made).

This method is intended for use in simple applications that could be simple command line applications that are going to be invoked periodiocally.

TsiCamera::Start

```
bool Start ();
```

This method starts an acquisition, but does not block.

The application may specify a callback function that will be called as each frame is completed and ready for retrieval using the ReadXxxxImage functions.

The application must call the StopCamera method before issuing another StartAcquistion call.

TsiCamera::Stop

```
bool Stop ();
```

This method is used to either cancel and / or clean up after an acquisition. This method must be called between subsequent calls to the StartCamera method.

TsiCamera::StartTriggerAcquisition

```
bool StartTriggerAcquisition (void)
```

This method configures the camera to acquire images based a trigger. The trigger can be software or hardware initiated. After this method is called, the camera will not deliver images until it has received a trigger.

This method does not block.

TsiCamera::StopTriggerAcquisition

```
bool StopTriggerAcquisition (bool rearm)
```

This method stops the camera from delivering trigger based images.

If the rearm parameter is false, the camera will not respond to any further triggers unless StartTriggerAcquisition is subsequently called.

If the rearm parameter is true, the camera will respond to the next received trigger. This can be useful when the host needs to stop the current acquisition and quickly configure the camera to respond to the next trigger stimulus without incurring the overhead of calling StartTriggerAcquisition.

If camera parameters are updated, this method must be called (and StartTriggerAcquisition subsequently called) prior to any further trigger based acquisitions.

This method does not block.

TsiCamera::SWTrigger

```
bool SWTrigger (void)
```

This method enables the host to generate a software based trigger.

TsiCamera::GetAcquisitionStatus

```
TSI_ACQ_STATUS_ID GetAcquisitionStatus ();
```

Returns the current status of the acquisition for applications that wish to poll for completion.

TsiCamera::GetExposeCount

```
int32 GetExposeCount ();
```

The GetExposeCount method returns the number of exposures that have been completed since a StartCamera call.

TsiCamera::GetFrameCount

```
int32 GetFrameCount ();
```

The GetFrameCount method returns the number of frames that have been completetly transferred to host memory since a StartCamera call.

TsiCamera::WaitForImage

```
bool WaitForImage ();
```

An application that wishes to poll for frame completion can call this function that will return TRUE when the next frame is available.

TsiCamera::ResetExposure

```
bool ResetExposure ();
```

This method can be used to reset the camera exposure to the default value based on its current configuration.

TsiCamera::GetErrorCode

```
TSI_ERROR_CODE GetErrorCode();
```

Returns the current error code for the camera, a returned value of zero indicates success. Clears any non-zero error code.

TsiCamera::ClearError

```
VOID ClearError ();
```

Clears any existing error code for the camera without having to read it using GetErrorCode.

TsiCamera::GetErrorStr

```
bool GetErrorStr(TSI_ERROR_CODE ErrorCode, char *StringBuffer,
int32 &StringLength)
```

Returns the non-localized ASCII string description of the supplied error code.

Passing a NULL pointer and pointer to a variable containing a length of zero length results in the length of the string associated with the supplied error code being returned so that a buffer of appropriate length can be dynamically allocated.

TsiCamera::GetLastErrorStr

```
char* GetLastErrorStr (void)
```

Returns a pointer to the non-localized ASCII string description of the current error code.

2.4.2. List of Camera Parameters

This list of parameters are used by the **TsiCamera.GetParameter** and **TsiCamera.SetParameter** methods.

TSI_PARAM_ATTR	**TSI_TYPE_UNS8:** For expert use only, contact us for more info.
TSI_PARAM_PROTOCOL	**TSI_TYPE_UNS32:** Obsolete
TSI_PARAM_FW_VER	**TSI_TYPE_TEXT:** Returns the firmware revision of the camera. The format of the returned data is MM.mm.rr.build, where MM is a two digit major version, mm is a two digit minor version, rr is a two digit revision, and build is a sequential number used internally by TSI supplied utilities.
TSI_PARAM_HW_VER	**TSI_TYPE_TEXT:** Returns the hardware rev of the camera. The format of the returned data is MM.mm.rr.build, where MM is a two digit major version, mm is a two digit minor version, rr is a two digit revision, and build is a sequential number used internally by TSI supplied utilities.
TSI_PARAM_HW_MODEL	**TSI_TYPE_TEXT:** Returns the hardware model name of the camera, format TBD.
TSI_PARAM_HW_SER_NUM	**TSI_TYPE_TEXT:** Returns the serial number of the camera, format TBD.
TSI_PARAM_CAMSTATE	**TSI_TYPE_UNS32 (enum):** Get the current state of the camera (IDLE / RUN)

TSI_PARAM_CAM_EXPOSURE_STATE	
TSI_PARAM_CAM_TRIGGER_STATE	**TSI_TYPE_UNS32 (enum):** Get the camera's current trigger state (NONE/ARMED/DISARMED/TRIGGERED)
TSI_PARAM_EXPOSURE_UNIT	**TSI_TYPE_UNS32 (enum):** Set the camera's exposure time unit of measurement.
TSI_PARAM_EXPOSURE_TIME	**TSI_TYPE_UNS32:** Set the camera's exposure time in the current unit of measurement. See TSI_PARAM_EXPOSURE_UNIT above. This parameter can be dynamically updated during image acquisition; however, the host application should wait for at least twice the current exposure time between consecutive updates to ensure that they are successfully applied.
TSI_PARAM_ACTUAL_EXPOSURE_TIME	**TSI_TYPE_UNS32:** Gets the actual exposure time in microseconds for a commanded exposure time.
TSI_PARAM_FRAME_TIME	**TSI_TYPE_UNS32:** Get the camera frame readout time in milliseconds.
TSI_PARAM_VSIZE	**TSI_TYPE_UNS32:** Get the camera sensor's number of active rows.
TSI_PARAM_HSIZE	**TSI_TYPE_UNS32:** Get the camera sensor's maximum number of active columns.
TSI_PARAM_ROI_BIN	**TSI_TYPE_UNS32 [6]:** Get/Set ROI and Binning Factors – ROI and binning are returned as an array of six UNS32 values, which can be encoded/decoded using the $$$ structure type defined above.
TSI_PARAM_FRAME_COUNT	**TSI_TYPE_UNS32:** Sets number of frames to acquire before going to the TSI_ACQ_DONE state.
TSI_PARAM_CURRENT_FRAME	**TSI_TYPE_UNS32:** Gets the number of frames completely read out in an "N" frame sequence.
TSI_PARAM_OP_MODE	**TSI_TYPE_UNS32 (enum):** Sets the current operating mode
TSI_PARAM_CDS_GAIN	**TSI_TYPE_UNS32 (enum):** Gets or Sets CDS gain setting
TSI_PARAM_VGA_GAIN	**TSI_TYPE_UNS32:** Gets or Sets VGA gain
TSI_PARAM_GAIN	**TSI_TYPE_UNS32:** Gets or Sets linear gain (x1 – xN)
TSI_PARAM_OPTICAL_BLACK_LEVEL	**TSI_TYPE_UNS32:** Sets the optical black level
TSI_PARAM_PIXEL_OFFSET	**TSI_TYPE_UNS32:** Obsolete
TSI_PARAM_READOUT_SPEED_INDEX	**TSI_TYPE_UNS32:** Selects the index into the set of readout speeds the camera supports.
TSI_PARAM_READOUT_SPEED	**TSI_TYPE_UNS32:** Returns the digitization rate in MHz for the given speed index.

TSI_PARAM_FRAME_RATE	**TSI_TYPE_UNS32:** Returns the frame rate in frames per second. Value is returned as one 32 bit quantity that is encoded as a single precision floating point number.
TSI_PARAM_COOLING_MODE	**TSI_TYPE_UNS32:** Sets the cooling mode. A value of 1 enables cooling and a value of 0 disables cooling. Not all camera models support cooling. A host application can ascertain support for this by checking the max value for this parameter (which will be 1 for a camera supporting cooling and 0 otherwise).
TSI_PARAM_COOLING_SETPOINT	**TSI_TYPE_INT32:** Sets the current CCD chamber temperature set point in hundredths of degrees centigrade (if supported by the camera). This value contains an implied decimal point, to convert to floating point degrees centegrade, divide by 100.0. Inversely, to convert from floating point degrees celcius, multiply that value by 100.0 and round or truncate (cast) as desired.
TSI_PARAM_TEMPERATURE	**TSI_TYPE_INT32: Not supported. Do not use.**
TSI_PARAM_QX_OPTION_MODE	**TSI_TYPE_UNS32 (enum):** Sets the QX mode
TSI_PARAM_TURBO_CODE_MODE	**TSI_TYPE_UNS32 (bool):** Sets TURBO Code Output On/Off
TSI_PARAM_XORIGIN	**TSI_TYPE_UNS32:** Gets or sets the X component of the upperleft corner of the specified ROI.
TSI_PARAM_YORIGIN	**TSI_TYPE_UNS32:** Gets or sets the Y component of the upperleft corner of the specified ROI.
TSI_PARAM_XPIXELS	**TSI_TYPE_UNS32:** Gets or sets the width in pixels of the desired ROI.
TSI_PARAM_YPIXELS	**TSI_TYPE_UNS32:** Gets or sets the height in pixels of the desired ROI.
TSI_PARAM_XBIN	**TSI_TYPE_UNS32:** Used to set or query the current horizontal binning factor.
TSI_PARAM_YBIN	**TSI_TYPE_UNS32:** Used to set or query the current horizontal binning factor.
TSI_PARAM_IMAGE_ACQUISITION_MODE	**TSI_TYPE_UNS32:** Obsolete, do not use
TSI_PARAM_NAMED_VALUE	**TSI_TYPE_TEXT:** For internal use only.
TSI_PARAM_TAPS_INDEX	**TSI_TYPE_UNS32:** Used to select or query the current tap setting.
TSI_PARAM_TAPS_VALUE	**TSI_TYPE_UNS32:** Used to query the number of taps for the current tap setting (See TSI_PARAM_TAPS_INDEX above).
TSI_PARAM_RESERVED_1	**Reserved for internal use.**
TSI_PARAM_RESERVED_2	**Reserved for internal use.**
TSI_PARAM_RESERVED_3	**Reserved for internal use.**

TSI_PARAM_RESERVED_4	**Reserved for internal use.**
TSI_PARAM_GLOBAL_CAMERA_NAME	**TSI_TYPE_TEXT:** Returns the camera name stored in the camera thats visible to users on the network and software loading the camera.
TSI_PARAM_CDS_GAIN_VALUE	**TSI_TYPE_UNS32:** For expert use only, contact us for details.
TSI_PARAM_PIXEL_SIZE	**TSI_TYPE_FP:** Size of the pixel width in microns (pixel assumed square).
TSI_PARAM_BITS_PER_PIXEL	**TSI_TYPE_UNS32:** Number of bits per pixel in raw data.
TSI_PARAM_BYTES_PER_PIXEL	**TSI_TYPE_UNS32:** Number of bytes per pixel (1 or 2).
TSI_PARAM_READOUT_TIME	**TSI_TYPE_FP:** Readout time of the camera in seconds.
TSI_PARAM_HW_TRIGGER_ACTIVE	**TSI_TYPE_UNS32:** Gets or sets the hardware active trigger state. 1 = Active, 0 = Not Active.
TSI_PARAM_HW_TRIG_SOURCE	**TSI_TYPE_UNS32 (enum):** Gets or sets the HW trigger source.
TSI_PARAM_HW_TRIG_POLARITY	**TSI_TYPE_UNS32 (enum):** Gets or sets the HW trigger polarity.
TSI_PARAM_TAP_BALANCE_ENABLE	**TSI_TYPE_UNS32 :** Gets or sets the tap balance enable. 1 = Active, 0 = Not Active.
TSI_PARAM_DROPPED_FRAMES	**TSI_TYPE_UNS32 :** Gets the number of dropped frames.
TSI_PARAM_EXPOSURE_TIME_US	**TSI_TYPE_UNS32 :** Gets or sets the exposure time in microseconds.
TSI_PARAM_RESERVED_5	**Reserved for internal use.**
TSI_PARAM_RESERVED_6	**Reserved for internal use.**
TSI_PARAM_RESERVED_7	**Reserved for internal use.**
TSI_PARAM_UPDATE_PARAMETERS	**TSI_TYPE_NONE :** Setting this parameter ensures that changes to the following parameters are applied to the camera: • TSI_PARAM_READOUT_SPEED • TSI_PARAM_TAPS_INDEX • TSI_PARAM_ROI_BIN • TSI_PARAM_TURBO_CODE_MODE It takes approximately 200 ms for the camera to complete the processing required for setting this parameter during which time it will be unavailable to respond to any commands or requests. This parameter is write-only, any attempt to read this parameter will fail – TsiCamera.GetParameter will return false.
TSI_PARAM_FEATURE_LIST	**TBD**

TSI_PARAM_FEATURE_VALID	**TBD**
TSI_PARAM_NUM_IMAGE_BUFFERS	**TSI_TYPE_INT32 :** Gets or sets the number of image buffers used for video frame acquisition. This parameter is optional and if it is not explicitly set, the default number of image buffers is 8. The aggregate size in memory of all the image buffers is a function of: • Image width W (can be read from the camera) • Image height H (can be read from the camera) • Bytes per pixel B (can be read from the camera) • The size of an internal implementation struct (S) which is fixed at 112 bytes. The equation for determining the total aggregate size in memory of the image buffers is: Memory size = ((W * H * B) + S) * (number of buffers)
TSI_PARAM_COLOR_FILTER_TYPE	**TSI_TYPE_INT8 (array):** Gets or sets the camera's color filter array type. This attribute will be set to some string other than "mono" for color enabled cameras. This is the basis for determining whether or not the camera supports color imaging.
TSI_PARAM_COLOR_FILTER_PHASE	This parameter is for expert use only. Please contact us for details.
TSI_PARAM_IR_FILTER_TYPE	This parameter is for expert use only. Please contact us for details.
TSI_PARAM_COLOR_CAMERA_ CORRECTION_MATRIX	This parameter is for expert use only. Please contact us for details.
TSI_PARAM_CCM_OUTPUT_ COLOR_SPACE	This parameter is for expert use only. Please contact us for details.
TSI_PARAM_DEFAULT_ WHITE_BALANCE_MATRIX	This parameter is for expert use only. Please contact us for details.
TSI_PARAM_USB_ENABLE_LED	**TSI_TYPE_UNS32 : Enables or disables the LEDs on the back of a USB camera. If this is set to 0, the LEDs will be turned OFF. If this is set to 1, the LEDs will be ON.**

2.5. TsiColorCamera Class

2.5.1. METHODS

TsiColorCamera::SetDemosaicFunction

```
bool SetDemosaicFunction (TSI_DEMOSAIC_FUNCTION f);
```

Specifies a user supplied function to demosaic the monochrome image.

NOTE: This has not been implemented – it will be implemented in a future TBD release.

TsiColorCamera::ConcatenateColorTransform

```
bool ConcatenateColorTransform (double* p3x3Matrix);
```

Specifies a user supplied 3x3 color transform matrix that will be applied to the demosaiced color pixel data during image acquisition.

The supplied matrix is concatenated with other specified matrices into a single matrix.

The default color transform matrix that is specified in the color processing pipeline is the identity matrix. If the host application needs to customize this behavior, it *must* call this method prior to the start of image acquisition.

TsiColorCamera::ConcatenateColorTransform

```
bool ConcatenateColorTransform (TSI_TRANSFORM t, unsigned int
outputBitDepth);
```

Specifies that an SDK supplied color transform matrix should be applied to the demosaiced color pixel data during image acquisition.

The supplied matrix is concatenated with other specified matrices into a single matrix.

The outputBitDepth argument can be used to specify the data width of the output pixel data. Since the internal pixel data is manipulated at the camera model specific bit width, it can be useful to specify a lower bit depth for the output data. For example, if the camera supplies data using 14 bits per pixel, this method can be used to specify that the output data is 8 bits per pixel. Specifying values greater than the native camera bit depth is undefined and not supported. The user can specify 0 for this argument which will enable the SDK to automatically choose the maximum bit depth based on the current camera configuration.

The default color transform matrix that is specified in the color processing pipeline is the identity matrix. If the host application needs to customize this behavior, it *must* call this method prior to the start of image acquisition.

TsiColorCamera::ClearColorPipeline

```
void ClearColorPipeline();
```

Clears the current color pipeline data structures and resets them to the default state.

The default state of the color pipeline is:

- Input transform: the identity function.
- Color transform matrix: the identity matrix.
- Output transform: the identity function.

If the host application wants to modify the current color pipeline, it must call this method prior to reconfiguring it. It is safe for the host application to call this method during image acquisition. Once the host has finished modifying the color pipeline, it must call FinalizeColorPipeline(). This will signal the SDK to use the new pipeline configuration on the "next" acquired frame subsequent to the call of FinalizeColorPipeline().

TsiColorCamera::SetInputTransform

```
bool SetInputTransform (int* pNx2pBMatrix, int N, int B);
```

Specifies a user supplied input transform LUT to be applied to the demosaiced color pixel data during image acquisition.

The N argument specifies the number of spectral channels which corresponds directly to the number of 1 dimensional LUTs that is supplied in the pNx2pBMatrix. For the current SDK release, this argument must be set to 3 since multispectral color processing is not currently supported. Support for multispectral color processing is scheduled for a future TBD release.

The B argument is the bit depth of the demosaiced color pixel data. This must be set to the native bit depth of the camera for the current SDK release. Different bit depths may be supported in a future TBD release.

TsiColorCamera::SetOutputTransform

```
bool SetOutputTransform (int* pMx2pBMatrix, int M, int B);
```

Specifies a user supplied input transform LUT to be applied to the demosaiced color pixel data during image acquisition.

The M argument specifies the number of spectral channels which corresponds directly to the number of 1 dimensional LUTs that is supplied in the pMx2pBMatrix. For the current SDK release, this argument must be set to 3 since multispectral color processing is not currently supported. Support for multispectral color processing is scheduled for a future TBD release.

The B argument is the bit depth of the demosaiced color pixel data. This must be set to the native bit depth of the camera for the current SDK release. Different bit depths may be supported in a future TBD release.

TsiColorCamera::FinalizeColorPipeline

```
bool FinalizeColorPipeline();
```

When the host application calls this method, it signals the SDK that it has completed configuring the color pipeline. The SDK will then apply the new pipeline configuration to the requested frame. It is safe to call this method during image acquisition since this will apply the pipeline to the "next" requested frame subsequent to the call.

TsiColorCamera::GetPendingColorImage

```
TsiColorImage* GetPendingColorImage(TSI_COLOR_PROCESSING_MODE
postProcess);
```

This method will return a pointer to the next available color image. The postProcess argument specifies the how color image should be processed prior to delivery to the requestor.

If the postProcess argument is set to TSI_COLOR_DEMOSAIC, then the SDK will demosaic the data only and return the color image.

If the postProcess argument is set to TSI_COLOR_POST_PROCESS, then the SDK will demosaic the data and apply the color pipeline to it prior to returning the color image to the requestor.

TsiColorCamera::GetLastPendingColorImage

```
TsiColorImage*
GetLastPendingColorImage(TSI_COLOR_PROCESSING_MODE postProcess);
```

This method will return a pointer to the last available color image. The postProcess argument specifies the how color image should be processed prior to delivery to the requestor.

If the postProcess argument is set to TSI_COLOR_DEMOSAIC, then the SDK will demosaic the data only and return the color image.

If the postProcess argument is set to TSI_COLOR_POST_PROCESS, then the SDK will demosaic the data and apply the color pipeline to it prior to returning the color image to the requestor.

TsiColorCamera::FreeColorImage

```
bool FreeColorImage();
```

This method frees the color image data object. The Close method and the destructor for this class will close and free up any image data objects that have been created. Any pointers to TsiColorImage objects that have been created must not be accessed after the object has been freed by the TsiCamera FreeColorImage or Close methods or the TsiSdk object's CloseSDK method or destructor.

2.6. TsiImage Class

2.6.1. DATA MEMBERS

```
unsigned int m_Width;              // Width  of image in pixels.
unsigned int m_Height;             // Height of image in pixels.
unsigned int m_BitsPerPixel;       // The number of significant
                                   //   bits per pixel in the
                                   //   pixel data.
unsigned int m_BytesPerPixel;      // The number of bytes
                                   //   consumed by a pixel.
unsigned int m_SizeInPixels;       // Size of image in pixels.
unsigned int m_SizeInBytes;        // Size of image in bytes.
unsigned int m_XBin;               // Horizontal binning value.
unsigned int m_VBin;               // Vertical binning value.
unsigned int m_ROI[4];             // The region of interest
                                   //  (subimage) that the pixels
                                   //  were gathered from.
                                   // Format: [x0, y0, x1, y1]
unsigned int m_ExposureTime_ms;    // Exposure time in
                                        milliseconds.
unsigned int m_FrameNumber;        // Frame number returned from
                                        frame grabber.

union
{
  void            *vptr;
  char             *i8;
  unsigned char    *ui8;
  short            *i16;
  unsigned short   *ui16;
  unsigned int     *ui32;

} m_PixelData;
```

2.6.2. METHODS

TsiImage::Copy

```
bool Copy(TsiImage *src)
```

Copies an existing TsiImage's data.

TsiImage::Clone

```
TsiImage* Clone(TsiImage *src)
```

Creates a new duplicate of the specified TsiImage.

2.7. TsiColorImage Class

2.7.1. DATA MEMBERS

```
unsigned int m_ColorImageSizeInBytes; // Size of color image in
bytes

union
{
  void            *vptr;
  char             *i8;
  unsigned char    *ui8;
  short            *i16;
  unsigned short   *ui16;
  uint32_t         *ui32;

   struct
   {
      unsigned char b;
      unsigned char g;
      unsigned char r;
   } *BGR_8;

   struct
   {
      unsigned short b;
      unsigned short g;
      unsigned short r;
   } *BGR_16;

   struct
   {
      uint32_t b;
      uint32_t g;
      uint32_t r;
   } *BGR_32;

} m_ColorPixelData;
```

Chapter 3 Thorlabs Worldwide Contacts

USA, Canada, and South America
Thorlabs, Inc.
56 Sparta Avenue
Newton, NJ 07860
USA
Tel: 973-300-3000
Fax: 973-300-3600
www.thorlabs.com
www.thorlabs.us (West Coast)
Email: sales@thorlabs.com
Support: techsupport@thorlabs.com

Europe
Thorlabs GmbH
Hans-Böckler-Str. 6
85221 Dachau
Germany
Tel: +49-(0)8131-5956-0
Fax: +49-(0)8131-5956-99
www.thorlabs.de
Email: europe@thorlabs.com

France
Thorlabs SAS
109, rue des Côtes
78600 Maisons-Laffitte
France
Tel: +33 (0) 970 444 844
Fax: +33 (0) 825 744 800
www.thorlabs.com
Email: sales.fr@thorlabs.com

Japan
Thorlabs Japan, Inc.
Higashi-Ikebukuro Q Building, 1F
2-23-2, Higashi-Ikebukuro,
Toshima-ku, Tokyo 170-0013
Japan
Tel: +81-3-5979-8889
Fax: +81-3-5979-7285
www.thorlabs.jp
Email: sales@thorlabs.jp

UK and Ireland
Thorlabs Ltd.
1 Saint Thomas Place, Ely
Cambridgeshire CB7 4EX
Great Britain
Tel: +44 (0)1353-654440
Fax: +44 (0)1353-654444
www.thorlabs.com
Email: sales.uk@thorlabs.com
Support: techsupport.uk@thorlabs.com

Scandinavia
Thorlabs Sweden AB
Bergfotsgatan 7
431 35 Mölndal
Sweden
Tel: +46-31-733-30-00
Fax: +46-31-703-40-45
www.thorlabs.com
Email: scandinavia@thorlabs.com

Brazil
Thorlabs Vendas de Fotônicos Ltda.
Rua Riachuelo, 171
São Carlos, SP 13560-110
Brazil
Tel: +55-16-3413 7062
Fax: +55-16-3413 7064
www.thorlabs.com
Email: brasil@thorlabs.com

China
Thorlabs China
Room A101, No. 100
Lane 2891, South Qilianshan Road
Putuo District
Shanghai
China
Tel: +86 (0) 21-60561122
Fax: +86 (0)21-32513480
www.thorlabschina.cn
Email: chinasales@thorlabs.com

THORLABS

www.thorlabs.com