chromasens
Imaging for Professionals

truePIXA | software integration manual

Contents

Preface

This document is the main source of reference for the Chromasens truePIXA family of multispectral acquisition devices. It outlines and details all steps and processes necessary for multispectral image acquisition and processing and supplies operating instructions for the Chromasens "Chromantis[1]" application[2].

Note: this manual only deals with the image capture, image post processing and color measurement aspect of the truePIXA device. As a member of the allPIXA family of line scan cameras, a vast array of camera hardware setup, configuration, control and calibration functions are accessible for the truePIXA through the CSAPI programming interface and CST software application, although many configuration settings available for the allPIXA must be left in their factory state for the truePIXA device. Where video capture functions are documented in this manual, it is assumed that the camera has been configured to actually emit video data in a format accepted by the currently employed capture hardware (frame grabber). In its factory state, the device is so configured.

[1] unofficial working title
[2] not contained in this preliminary document

1

Principles of operation

The Chromasens truePIXA imaging device, in conjunction with a matching and calibrated illumination module, allows you to capture multichannel and spectral images as well as images in reference color spaces.

This section describes the principles of operation in a synoptic fashion and in general terms rather than by code examples and function references.

1.1 Definitions

This section defines some of the important terms used throughout this document.

band while for RGB or other color triplets, any of the three components is often referred to as a *color channel*, the individual components of a multichannel image or spectral image are referred to as *bands*.

raw image raw image data is transferred from the truePIXA camera into the main memory of the target PC by the capture hardware (frame grabber). From the application's point of view, the raw image is a normal RGB image with a sample depth of 8 or 16 bits per channel. The raw image cannot be used directly; must be transformed by the appropriate API functions first.

multichannel image raster image data with a number of samples per pixel (bands) that is larger than three. The colorimetric meaning of each channel is not necessarily well-defined. Like the color space of an uncalibrated RGB camera, the samples span an arbitrary color space unrelated to any standard space.

standard space image output image data can be configured to contain samples in the CIE L*a*b* or CIE XYZ color spaces.

spectral image a multichannel image in which the samples of each pixel represent linear relative spectral intensity in a given wavelength range. Typically, the wavelength interval covered by each sample is 10 nm.[1]

capture hardware hardware component capable of transferring raw video data from the truePIXA camera to a computer for further processing. Currently, this is essentially synonymous with a PCIe CameraLink frame grabber, as this is the only supported data path for the truePIXA camera.

geometry calibration the particular scan principle of the truePIXA camera requires a geometry calibration step for each scan setup. This means that once a truePIXA device and the corresponding illumination module have been set up in a scanning environment and the position of the camera relative to the target is finalized, a calibration target must be scanned and made available to the API.

[1] Currently 10 nm, range: 380-730 nm; 36 samples.

geometry transformation (image registration) using the data produced in the geometry transformation step, the API modifies the image pixel data from the truePIXA device such that all 12 samples of each of the multichannel pixels are exactly super-positioned. Effectively, this means registering the gray level images representing the 12 multichannel image bands.

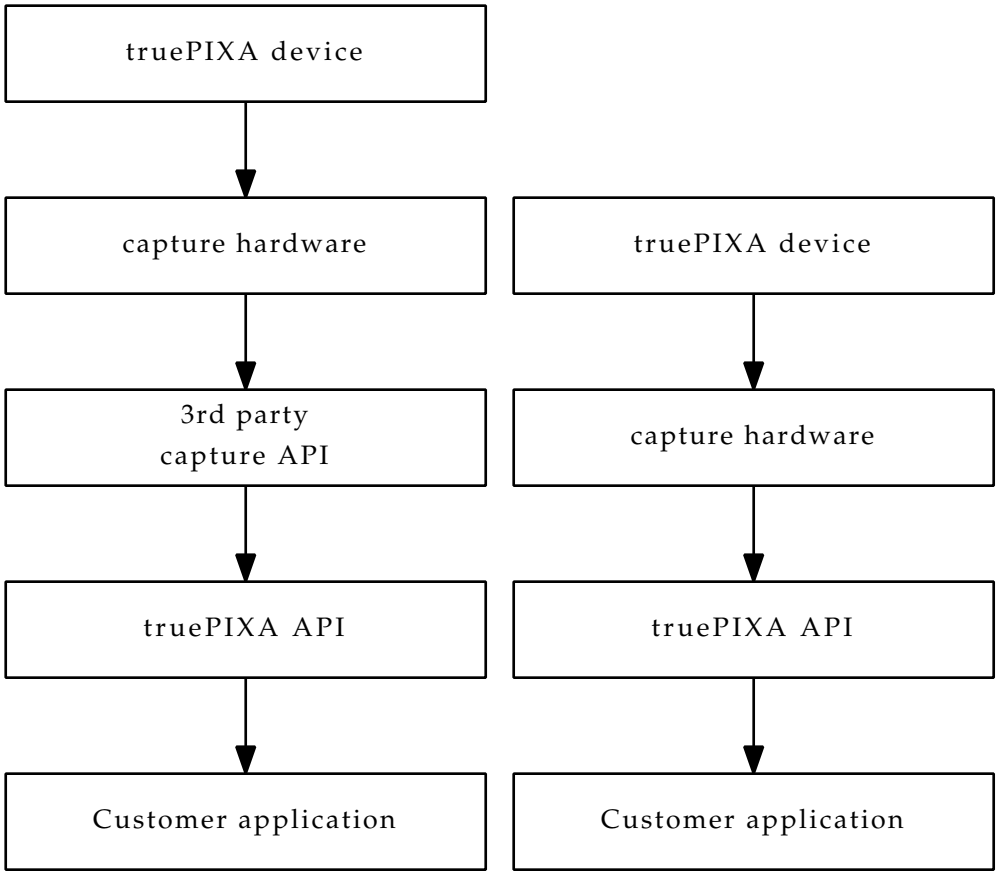
result descriptor the image processing functions compute various result images from the camera or raw source input. By specifying a result descriptor, the user application can specify a number of regions of interest and a corresponding result type.

1.2 Handling image capture and calibration

Due to the unique operational principle of the device, image capture with the truePIXA camera is slightly more complex than with a standard monochrome or RGB capture setup.

The raw image information captured using conventional capture hardware must be post processed before it can be incorporated into the customer's application for display, analysis or storage.

The Chromasens truePIXA Application Programming Interface allows for two basic paths of image capture.



(a) capture using the 3rd party capture API (b) direct capture from the truePIXA API

Figure 1.1: paths for image capture

1.2.1 Geometry calibration

The geometric calibration is currently a simple registration process which spatially matches the 12 color bands of the device. To carry out the calibration, a complete frame must be acquired in the final scan mode (speed and direction) and the resulting image must be passed to the API calibration function.

The function returns a handle to a transformation which can subsequently be used to either compute result images from raw input or to serialize the transformation and store it in a file on disk for later use.

Figure 1.2: Calibration target 03

Figure 1.2 shows the only currently supported calibration target. Each of the markers is detected and identified in all of the 12 sub-band images. For the calibration to yield results of sufficient accuracy, the target must cover the full scan width without being cropped by the image borders (i. e. all markers must be identifiable). The printed target is available in a number of sizes; pick the size that just fits within the scan area without cropping and without leaving too much white space near the image borders.

1.2.2 Independent capture

Independent capture in this context means that the caller side is responsible for capturing the raw image information required to produce multichannel or multispectral output from the raw output image emitted by the truePIXA camera.

Raw images can be captured from a truePIXA by a device chosen and operated by the customer. In this first step, the truePIXA device appears just like an ordinary trilinear line scan camera such as the allPIXA.

1.2.3 Integrated capture

When using the integrated capture facility of the truePIXA API, the API itself controls the capture hardware through one of several capture modules [2].

The advantage of this capture approach, as seen from the customer's perspective, is that the truePIXA camera appears as a monolithic source of image information and the need to integrate at least two different programming interfaces disappears. On the other hand, it may then be more difficult or even impossible to handle more complex scan scenarios such as variable transport speeds.

1.3 Image data access and image file format

In the low-level "C" interface, images, like other objects manipulated by API calls, are referenced by *handles*. A handle is simply an arbitrary and opaque integer value. Transfer of image data to and from API functions is controlled by such handles. API functions used for obtaining image meta data (such as size and resolution) expect a handle as input or return a handle. The following rules currently apply to the internal organization of image data:

- All raster data accessed and generated by the API is represented in raster scan order from left to right, top to bottom
- Bands and channels must be interleaved. Planed image organization is not supported.
- Input images may have a *stride* (in bytes) that differs from the value

$$stride_{unpadded} = \frac{width \times numberofsamples \times bitspersample}{8} \quad bitspersample \in \{8, 16, 32\}$$

- The stride of output images is always equal to $stride_{unpadded}$.
- The TIFF file format is the only supported file format.

[2] This preliminary version does not contain any modules for integrated capture

2

Platform support

(chapter is incomplete)

2.1 Supported operating systems (PRELIMINARY)

Microsoft Windows XP x64

Microsoft Windows Vista x64

Microsoft Windows 7 x64

Microsoft Windows 8 x64

2.2 Language and development platform bindings (PRELIMINARY)

C native: DLL module `tpapi.dll`, headers and library

C++ native: headers and source files

C# CLI / .NET: DLL module / type library

3

Basic “C” API usage

3.1 The C language API

The C language interface is provided for maximum compatibility. All other language bindings make direct use of the “C” functions exposed by `tpapi.dll`.

The API tries to avoid the usage of structures and pointers. In lieu of structure types and pointers thereto, opaque handles are used throughout the API. Many functions require a handle as an argument or return a handle to an object. Object properties can be queried or set by corresponding *set* and *get* functions.

A handle is simply a 32-bit integer value represented by the type `TRPI_Handle`.

This chapter briefly outlines how to use the basic “C” language API to access truePIXA functionality.

To obtain spectral measurements with independent capture, these steps are generally required:

1. Initialization
2. restoring a geometry transformation handle
3. capturing the image using the existing method
4. Assigning the image to the API by creating a new API image
5. Defining a job descriptor
6. Triggering the transformation process
7. Obtaining the result images and working with measurement results.

3.2 Initialization

Upon customer application start-up, the global function `TRPI_Initialize()` must be called to initialize the API.

3.3 Restoring a geometry transformation handle

This example assumes that geometry transformation has been calibrated and the calibration data has been serialized. It can thus be loaded from disk using standard library functions, and be passed to the API by calling `TRPI_deSerializeTranform`. The serialized data is standard UTF8 (ASCII range, null-terminated string, no multibyte characters).

3.4 Capturing the image using the existing method

This simple example assumes that the user has connected the camera and established a method to capture frames.

3.5 Assigning the image to the API by creating a new API image

Once a frame has been captured, call `TRPI_createNewImage()` with the appropriate meta data and a pointer to your raw image data. If the image data produced by your capturing process is stored in an organization other than sample-interleaved, it is necessary to rearrange the data into an interleaved representation before calling `TRPI_createNewImage`. With the data pointer being non-NULL, the image data will not be copied but referenced and the memory required by the API representation of the image is minimal. Still, the image object must be freed by calling `TRPI_DeleteObject()` when no longer needed.

3.6 Defining a job descriptor

A job descriptor is a list of regions and image types that you would like to be computed from a scanned raw frame. In this basic example, we'd like to compute a full spectral image and a full L*a*b* image from the scanned frame. The API calls necessary to achieve this are `TRPI_CreateJobDescriptor()` and `TRPI_JobDescriptorAddRequest()`. Since we want the full image, we can either use the image geometry known through the scan process or call `TRPI_GetImageWidth()` and `TRPI_GetImageHeight()` using the handle created by `TRPI_createNewImage()`.

3.7 Triggering the transformation process

The last step is to start the actual computation by calling `TRPI_TransformImage()` with the handles for raw image, transformation and job descriptor.

3.8 Obtaining the result images and working with measurement results

After the call to `TRPI_TransformImage()` returns without error, result images can be queried with the `TRPI_JobDescriptorGetResult()` function, which returns an image handle, and the functions for accessing image meta data such as `TRPI_GetImageWidth()` and `TRPI_GetImageDataPointer()`. When done, all images created during the computation can be removed by calling `TRPI_DeleteObject()` and passing the job descriptor handle as input.

3.9 Example code

The following listing collates all previous steps into a single code example.

```
1  #include <tpapi.h>
2  #include <stdio.h>
3
4  void main()
5  {
6      /* NOTE: no error handling is performed
7         in this example */
8
9      TRPI_Handle myTransform=0;
10     TRPI_Handle myRawImage=0;
11     TRPI_Handle myDescriptor=0;
12     TRPI_Handle resultLab=0;
13     TRPI_Handle resultSpectral=0;
14     TRPI_Int idLab=0, idSpect=0;
15
16     /* Initialize API */
17     TRPI_Initialize();
18
19     /* access to previously created and stored
20        transformation data */
21     char const* const transData = LoadTransformString();
22                                   /* user provided function */
23
24     /* create transform handle */
25     myTransform = TRPI_DeSerializeTransform(transData);
26
27     /* scan image */
28     /* ... */
29
30     /* create API image */
31     myRawImage = TRPI_CreateNewImage(width,  /* user supplied */
```

```
32                                      height, /* user supplied */
33                                      stride,  /* user supplied */
34                                      TRPI_FormatU8,
35                                      3, TRPI_RGB, 100, 100,
36                                      data /* user supplied */);
37
38
39      /* create descriptor and add requests */
40      myDescriptor = TRPI_CreateJobDescriptor();
41      idLab = TRPI_JobDescriptorAddRequest(myDescriptor,
42                              TRPI_TypeLab,
43                              0, 0,
44                              TRPI_GetImageWidth(myRawImage)-1,
45                              TRPI_GetImageHeight(myRawImage)-1);
46
47      idSpect = TRPI_JobDescriptorAddRequest(myDescriptor,
48                              TRPI_TypeSpectral,
49                              0, 0,
50                              TRPI_GetImageWidth(myRawImage)-1,
51                              TRPI_GetImageHeight(myRawImage)-1);
52
53      /* compute output images */
54      TRPI_TransformImage(myRawImage, myTransform, myDescriptor);
55
56      /* save results */
57      TRPI_SaveImage(TRPI_JobDescriptorGetResult(idLab), "lab.tif");
58      TRPI_SaveImage(TRPI_JobDescriptorGetResult(idSpect), "spectral.tif");
59
60      /* clean up API objects */
61      TRPI_DeleteObject(myTransform);
62      TRPI_DeleteObject(myDescriptor);
63      TRPI_DeleteObject(myRawImage);
64
65      /* ... */
66  }
```

3.10 Measuring arbitrarily shaped regions

By making use of the `TRPI_JobDescriptorAddRequest()` function, it is possible to request any number of rectangular regions in an image for spectral measurement. Each of the regions so requested is computed in the form of an image with a format required by the API. While care is taken by the underlying API functions to make use of intersections between the ROI rectangles for optimization, i.e. the function `TRPI_TranformImage` organizes the regions and the sequence of computation such as to avoid redundancy, the process of transforming entire rectangular image sections is still *computationally expensive*. The main reason for this expensiveness is the fact that every pixel value in the result images is routed through a complex pipeline of computations that involve a full spectral reconstruction and hence the handling of a complete spectrum for each individual pixel.

This kind of computation is not always necessary. In many cases, it is sufficient to compute a single spectral value or CIEL*a*b* result for a given image region. While this can be achieved by calling `TRPI_JobDescriptorGetResultAverage()`, the regions specified in this way are limited to rectangles.

By using the API function `TRPI_MeasureLabeledComponents()`, it is possible to achieve a drastic speed up for multi-region measurements. The function expects three parameters:

1. `LCI` This is the handle to the "labeled component image", an RGB image in which the pixel values represent an ID of the region to which they belong.

2. `MC` A handle to the multichannel image on which the measurement will be based. This image can be produced by using the `TRPI_TranformImage` function; see below for details.

3. `numResults` receives the number of results produced; this value is normally identical to the number of regions that were found within the image referred to by `LCI`.

4

API reference

4.1 Data Structure Index

4.1.1 Data Structures

Here are the data structures with brief descriptions:

4.2 File Index

4.2.1 File List

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

4.3 Data Structure Documentation

4.3.1 TRPI_MultiMeasurementResult Struct Reference

`#include <tpapi.h>`

Data Fields

- unsigned int **ID**
- float **Lab** [3]
- float **XYZ** [3]
- float **Spectrum** [36]

4.3.1.1 Detailed Description

Structure holding measurement results in Lab, XYZ and spectrum. Used for new labeled component images.

4.3.1.2 Field Documentation

4.3.1.2.1 unsigned int ID

holds the ID of the segment for which this value is valid

4.3.1.2.2 float Lab[3]

L∗a∗b∗ result value

4.3.1.2.3 float Spectrum[36]

result spectrum

4.3.1.2.4 float XYZ[3]

XYZ result value

The documentation for this struct was generated from the following file:

- **tpapi.h**

4.4 File Documentation

4.4.1 tpapi.h File Reference

Data Structures

- struct **TRPI_MultiMeasurementResult**

Macros

- #define **TRPIAPI** __declspec(dllimport)
- #define **TRPI_InvalidHandle** 0x0

Typedefs

- typedef unsigned __int32 **TRPI_Handle**
- typedef __int32 **TRPI_Int**
- typedef unsigned __int32 **TRPI_UInt**
- typedef __int16 **TRPI_Short**
- typedef unsigned __int16 **TRPI_UShort**
- typedef __int8 **TRPI_Byte**
- typedef unsigned __int8 **TRPI_UByte**
- typedef float **TRPI_Float**
- typedef __int32 **TRPI_Error**
- typedef void(__stdcall * **TRPI_ProgressCallbackFunction**)(**TRPI_Handle** object, float progress)

Enumerations

- enum **TRPI_CaptureState** {
  TRPI_CaptureNone, **TRPI_CaptureNotInitialized**, **TRPI_CaptureIdle**, **TRPI_CaptureCapturing**,
  TRPI_CaptureError }
- enum {
  TRPI_UnknownError = -1, **TRPI_Ok**, **TRPI_BugCheck**, **TRPI_OperationNotSupported**,
  TRPI_UnknownHandle, **TRPI_UnknownTransformationHandle**, **TRPI_UnknownImageHandle**, **TRPI_UnknownJobHandle**,
  TRPI_UnknownParameter, **TRPI_InvalidData**, **TRPI_InvalidParameter**, **TRPI_InvalidRequest**,
  TRPI_InvalidROI, **TRPI_ImageNotComputed**, **TRPI_FileNotFound**, **TRPI_FileAccessError**,
  TRPI_FileFormatError, **TRPI_CalibrationError**, **TRPI_ImageProcessingError** }
- enum **TRPI_ImageType** {
  TRPI_TypeRGB, **TRPI_TypeSRGB**, **TRPI_TypeMultichannel**, **TRPI_TypeXYZ**,
  TRPI_TypeLab, **TRPI_TypeSpectral** }
- enum **TRPI_SampleFormat** {
  TRPI_FormatU8, **TRPI_FormatS8**, **TRPI_FormatU16**, **TRPI_FormatS16**,
  TRPI_FormatU32, **TRPI_FormatS32**, **TRPI_FormatFloat** }
- enum **TRPI_SampleSequence** { **TRPI_RGBx**, **TRPI_BGRx** }

Functions

- TRPIAPI **TRPI_Error TRPI_Initialize** ()
- TRPIAPI **TRPI_Error TRPI_GetLastError** ()
- TRPIAPI **TRPI_Error TRPI_DeleteObject** (**TRPI_Handle** handle)
- TRPIAPI char const * **TRPI_TranslateError** (**TRPI_Error** error)
- TRPIAPI **TRPI_Int TRPI_GetNumCaptureSources** ()
- TRPIAPI char const * **TRPI_GetCaptureSourceName** (**TRPI_Int** num)
- TRPIAPI **TRPI_Handle TRPI_CreateCaptureSource** (char const *name)
- TRPIAPI **TRPI_Handle TRPI_CreateCaptureSourceByIndex** (**TRPI_Int** index)
- TRPIAPI **TRPI_Error TRPI_ConfigureCaptureSource** (**TRPI_Handle** source, char const *xml_configuration)
- TRPIAPI char const * **TRPI_GetCaptureSourceConfiguration** (**TRPI_Handle** source)
- TRPIAPI void * **TRPI_QueryCaptureSourceProperty** (**TRPI_Handle** source, char const *value_name, **TRPI_Error** *result)
- TRPIAPI **TRPI_Error TRPI_SetCaptureSourceProperty** (**TRPI_Handle** source, char const *value_name, void *new_value)
- TRPIAPI **TRPI_CaptureState TRPI_GetCaptureStatus** (**TRPI_Handle** source)
- TRPIAPI **TRPI_Error TRPI_BeginCapture** (**TRPI_Handle** source, **TRPI_Handle** transformation, **TRPI_Handle** jobDescriptor)
- TRPIAPI **TRPI_Error TRPI_StopCapture** (**TRPI_Handle** source)
- TRPIAPI **TRPI_Error TRPI_SetCaptureCallback** (**TRPI_Handle** source, **TRPI_ProgressCallbackFunction** callback)
- TRPIAPI **TRPI_Handle TRPI_GetSourceWorkImage** (**TRPI_Handle** source)
- TRPIAPI **TRPI_Handle TRPI_CreateNewImage** (**TRPI_Int** width, **TRPI_Int** height, **TRPI_Int** stride, **TRPI_SampleFormat** sample_format, **TRPI_Int** num_channels, **TRPI_ImageType** image_type, **TRPI_Float** resolutionX, **TRPI_Float** resolutionY, void *data_pointer)
- TRPIAPI **TRPI_Handle TRPI_CreateNewImageEx** (**TRPI_Int** width, **TRPI_Int** height, **TRPI_Int** stride, **TRPI_SampleFormat** sample_format, **TRPI_SampleSequence** sequence, **TRPI_Int** num_channels, **TRPI_ImageType** image_type, **TRPI_Float** resolutionX, **TRPI_Float** resolutionY, void *data_pointer)
- TRPIAPI **TRPI_Error TRPI_SaveImage** (**TRPI_Handle** image, char const *file_name)
- TRPIAPI **TRPI_Handle TRPI_LoadImage** (char const *file_name)
- TRPIAPI **TRPI_Error TRPI_GetImageWidth** (**TRPI_Handle** image, **TRPI_Int** *value)
- TRPIAPI **TRPI_Error TRPI_GetImageHeight** (**TRPI_Handle** image, **TRPI_Int** *value)
- TRPIAPI **TRPI_Error TRPI_GetImageStride** (**TRPI_Handle** image, **TRPI_Int** *value)
- TRPIAPI **TRPI_Error TRPI_GetImageSampleFormat** (**TRPI_Handle** image, **TRPI_SampleFormat** *value)
- TRPIAPI **TRPI_Error TRPI_GetImageChannelNumber** (**TRPI_Handle** image, **TRPI_Int** *value)
- TRPIAPI **TRPI_Error TRPI_GetImageType** (**TRPI_Handle** image, **TRPI_ImageType** *value)
- TRPIAPI **TRPI_Error TRPI_GetImageDataPointer** (**TRPI_Handle** image, void **pointer)
- TRPIAPI **TRPI_Error TRPI_GetImageResolutionX** (**TRPI_Handle** image, **TRPI_Float** *value)
- TRPIAPI **TRPI_Error TRPI_GetImageResolutionY** (**TRPI_Handle** image, **TRPI_Float** *value)
- TRPIAPI **TRPI_Error TRPI_InitWhiteRefCache** (**TRPI_Handle** image, **TRPI_Int** use_configuration, **TRPI_Int** first_row, **TRPI_Int** last_row)
- TRPIAPI **TRPI_Handle TRPI_CalibrateTransform** (**TRPI_Handle** raw_input_image, **TRPI_Error** *result)
- TRPIAPI **TRPI_Handle TRPI_CalibrateTransformEx** (**TRPI_Handle** input_image, unsigned int flags, **TRPI_Error** *result)
- TRPIAPI char const * **TRPI_SerializeTransform** (**TRPI_Handle** transformation)
- TRPIAPI **TRPI_Handle TRPI_DeSerializeTransform** (char const *str)
- TRPIAPI **TRPI_Error TRPI_GetTransformNetImageSize** (**TRPI_Handle** transformation, **TRPI_Int** *maxWidth, **TRPI_Int** *maxHeight)
- TRPIAPI **TRPI_Handle TRPI_TransformToFastRGB** (**TRPI_Handle** transformation, **TRPI_Handle** raw_image, **TRPI_Handle** result_image)
- TRPIAPI **TRPI_Error TRPI_TransformImage** (**TRPI_Handle** raw_image, **TRPI_Handle** transformation, **TRPI_Handle** job_descriptor)

- TRPIAPI **TRPI_Handle TRPI_CreateJobDescriptor** ()
- TRPIAPI **TRPI_Int TRPI_JobDescriptorAddRequest** (**TRPI_Handle** descriptor, **TRPI_ImageType** type, **TRPI_Int** roiStartX, **TRPI_Int** roiStartY, **TRPI_Int** roiEndX, **TRPI_Int** roiEndY)
- TRPIAPI **TRPI_Int TRPI_JobDescriptorAddRequestEx** (**TRPI_Handle** descriptor, **TRPI_ImageType** type, **TRPI_SampleFormat** format, **TRPI_Int** roiStartX, **TRPI_Int** roiStartY, **TRPI_Int** roiEndX, **TRPI_Int** roiEndY)
- TRPIAPI **TRPI_Handle TRPI_JobDescriptorGetResult** (**TRPI_Handle** descriptor, **TRPI_Int** id)
- TRPIAPI void * **TRPI_JobDescriptorGetResultAverage** (**TRPI_Handle** descriptor, **TRPI_Int** id, **TRPI_Int** *numValues)
- TRPIAPI **TRPI_Error TRPI_SetConfigurationString** (char const *par_name, char const *par_value)
- TRPIAPI **TRPI_Error TRPI_SetConfigurationInt** (char const *par_name, int par_value)
- TRPIAPI **TRPI_Error TRPI_SetConfigurationFloat** (char const *par_name, **TRPI_Float** par_value)
- TRPIAPI **TRPI_Error TRPI_GetConfigurationString** (char const *par_name, char const **v)
- TRPIAPI **TRPI_Error TRPI_GetConfigurationInt** (char const *par_name, **TRPI_Int** *v)
- TRPIAPI **TRPI_Error TRPI_GetConfigurationFloat** (char const *par_name, **TRPI_Float** *v)
- TRPIAPI **TRPI_Error TRPI_LoadConfiguration** (char const *xml_file_name)
- TRPIAPI **TRPI_Error TRPI_SaveConfiguration** (char const *xml_file_name)
- TRPIAPI void **TRPI_SpectrumToXYZ** (float *sp, double *xyz)
- TRPIAPI void **TRPI_XYZTOLab50** (double X, double Y, double Z, float *L, float *a, float *b)
- TRPIAPI void * **TRPIPRIVATE** (int index)
- TRPIAPI
  TRPI_MultiMeasurementResult
  const * **TRPI_MeasureLabeledComponents** (**TRPI_Handle** LCI, **TRPI_Handle** MC, int *numResults)
- TRPIAPI
  TRPI_MultiMeasurementResult
  const * **TRPI_MultiMeasurementResultEx** (**TRPI_Int** LCIwidth, **TRPI_Int** LCIheight, **TRPI_Int** LCIstride, void *LCIdata_pointer, **TRPI_Handle** MC, int *numResults)

4.4.1.1 Detailed Description

language interface for the truePIXA API

4.4.1.2 Macro Definition Documentation

4.4.1.2.1 #define TRPI_InvalidHandle 0x0

textual representation of an invalid handle value (0)

4.4.1.3 Typedef Documentation

4.4.1.3.1 typedef __int8 TRPI_Byte

generic 8bit integer

4.4.1.3.2 typedef __int32 TRPI_Error

error return type

4.4.1.3.3 typedef float TRPI_Float

standard 32-bit floating point

4.4.1.3.4 typedef unsigned __int32 TRPI_Handle

Handle type for reference to objects (Images, Transformations, Jobs)

4.4.1.3.5 typedef __int32 TRPI_Int

generic 32bit integer

4.4.1.3.6 typedef void(__stdcall * TRPI_ProgressCallbackFunction)(TRPI_Handle object, float progress)

callback function type

A callback function can be associated with a capture source or passed to a processing function.

Parameters

object	contains the handle to an image or a source or NULL, depending on how the callback pointer was used.
progress	float value indicating the progress of an operation, scaled from 0.0 to 1.0. Value 1.0 also indicates completion.

4.4.1.3.7 typedef __int16 TRPI_Short

generic 16bit integer

4.4.1.3.8 typedef unsigned __int8 TRPI_UByte

generic 8bit unsigned integer

4.4.1.3.9 typedef unsigned __int32 TRPI_UInt

generic 32bit unsigned integer

4.4.1.3.10 typedef unsigned __int16 TRPI_UShort

generic 16bit unsigned integer

4.4.1.4 Enumeration Type Documentation

4.4.1.4.1 enum TRPI_CaptureState

Lists all possible capture states

Enumerator

TRPI_CaptureNone capture status cannot be determined

TRPI_CaptureNotInitialized Source was not initialized; this will not normally be returned as sources are initialized upon creation, but a source may have no default configuration and require some parameters to be set before it can be set up

TRPI_CaptureIdle Source is idle and ready to capture

TRPI_CaptureCapturing Source is currently acquiring image data

TRPI_CaptureError Source is in an error state

4.4.1.4.2 enum TRPI_ImageType

Enumeration for all supported image types

Enumerator

TRPI_TypeRGB arbitrary (user) RGB (3 channels)

TRPI_TypeSRGB sRGB (3 channels)

TRPI_TypeMultichannel Multichannel (typically 12 channels)

TRPI_TypeXYZ XYZ representation (3 channels)

TRPI_TypeLab LAB encoding (3 channels)

TRPI_TypeSpectral spectral image (typically 36 channels)

4.4.1.4.3 enum TRPI_SampleFormat

Enumeration for all supported sample formats

Enumerator

TRPI_FormatU8 8 bits unsigned

TRPI_FormatS8 8 bits signed

TRPI_FormatU16 16 bits unsigned

TRPI_FormatS16 16 bits signed

TRPI_FormatU32 32 bits unsigned

TRPI_FormatS32 32 bits signed

TRPI_FormatFloat 32 bits floating point

4.4.1.4.4 enum TRPI_SampleSequence

sequence of channels within a pixel

Enumerator

TRPI_RGBx RGB with fourth channel

TRPI_BGRx BGR with fourth channel

4.4.1.5 Function Documentation

4.4.1.5.1 TRPIAPI TRPI_Error TRPI_BeginCapture (TRPI_Handle *source*, TRPI_Handle *transformation*, TRPI_Handle *jobDescriptor*)

starts image capture with the indicated source

Parameters

source	Handle to capture source
transformation	if non-null, expected to be a valid handle to a previously created or restored geometry transformation. In that case, jobDescriptor must also be valid and refer to a valid job descriptor.
jobDescriptor	if the transformation parameter is non-null, must refer to a non-empty job descriptor.

4.4.1.5.2 TRPIAPI TRPI_Handle TRPI_CalibrateTransform (TRPI_Handle *raw_input_image*, TRPI_Error * *result*)

Calibrate the geometry transform.

truePIXA output images must be postprocessed before they can be used by the customer application. One part of postprocessing is the geometry transformation. Geometry transformation produces a multichannel image with square pixels from raw input. In order to do that, it must know about the camera's physical location relative to the scan target. This information is obtained during geometry calibration.

This function analyzes the scan of a known image and derives a transformation from that analysis. The transformation is returned as a handle. The image must be a scan of one of the supported calibration targets; the target type is determined automatically. It is vitally important that the resolution in DPI of the raw_input_image image object has been set correctly for both dimensions. Failure to set the correct resolution value or at least a value close to the actual resolution will cause the calibration to fail.

Parameters

raw_input_image	handle to a raw image, as scanned by a capture source.
result	contains a result status

Returns

upon success, handle to the transformation. The transformation is bound to and only valid for raw images of the same dimensions as raw_input_image.

4.4.1.5.3 TRPIAPI TRPI_Error TRPI_ConfigureCaptureSource (TRPI_Handle *source*, char const * *xml_configuration*)

sets the complete configuration for a capture source

Each capture source has a number of configuration values (property/value pairs) that determine its internal configuration. The exact number, names and meaning of this configuration depends on the source.

Parameters

source	Handle to capture source
xml_configuration	NULL-terminated string formatted as a valid XML document, containing parameter/value entries valid for this source.

4.4.1.5.4 TRPIAPI TRPI_Handle TRPI_CreateCaptureSource (char const * *name*)

create a new capture source

Parameters

name	name of the new source

Returns

a handle to the newly created source, or TRPI_InvalidHandle.

4.4.1.5.5 TRPIAPI TRPI_Handle TRPI_CreateCaptureSourceByIndex (TRPI_Int *index*)

create a new capture source by numerical index

Parameters

index	value between 0 and the return value of TRPI_getNumCaptureSources, indicating the index of the source to create an instance of

4.4.1.5.6 TRPIAPI TRPI_Handle TRPI_CreateJobDescriptor ( )

creates a new empty job descriptor

A job descriptor is a list of ROI's and target types to be captured from a source or computed from an existing raw image.

4.4.1.5.7 TRPIAPI TRPI_Handle TRPI_CreateNewImage (TRPI_Int *width*, TRPI_Int *height*, TRPI_Int *stride*, TRPI_SampleFormat *sample_format*, TRPI_Int *num_channels*, TRPI_ImageType *image_type*, TRPI_Float *resolutionX*, TRPI_Float *resolutionY*, void * *data_pointer*)

creates a new image

This function creates a new API image.

Parameters

width	Image width
height	Image height
stride	Image stride (byte distance from scanline to scanline)
sample_format	input sample format, one of the TRPI_Format{..} constants.
num_channels	number of image channels
image_type	a supported image type
resolutionX	the resolution in dots per inch in CCD direction
resolutionY	the resolution in dots per inch in transport direction
data_pointer	if non-NULL, must point to a user supplied memory memory block of correct size (at least height∗stride bytes). The newly created image will then reference user data. The user must ensure that this data remains valid until the image is deleted with the TRPI_DeleteObject function. If NULL, the function allocates the required space.

4.4.1.5.8 TRPIAPI TRPI_Handle TRPI_CreateNewImageEx (TRPI_Int *width*, TRPI_Int *height*, TRPI_Int *stride*, TRPI_SampleFormat *sample_format*, TRPI_SampleSequence *sequence*, TRPI_Int *num_channels*, TRPI_ImageType *image_type*, TRPI_Float *resolutionX*, TRPI_Float *resolutionY*, void * *data_pointer*)

creates a new image

This function creates a new API image and allows some extra parameters to be passed in comparison to TRPI_CreateNewImage.

Parameters

width	Image width
height	Image height
stride	Image stride (byte distance from scanline to scanline)
sample_format	input sample format, one of the TRPI_Format{..} constants.
sequence	indicates the sequence of color channels
num_channels	number of image channels
image_type	a supported image type
resolutionX	the resolution in dots per inch in CCD direction
resolutionY	the resolution in dots per inch in transport direction
data_pointer	if non-NULL, must point to a user supplied memory memory block of correct size (at least height∗stride bytes). The newly created image will then reference user data. The user must ensure that this data remains valid until the image is deleted with the TRPI_DeleteObject function. If NULL, the function allocates the required space.

4.4.1.5.9 TRPIAPI TRPI_Error TRPI_DeleteObject (TRPI_Handle *handle*)

deletes an API object

This function deletes the image, transformation or job descriptor referred to by handle.

4.4.1.5.10 TRPIAPI TRPI_Handle TRPI_DeSerializeTransform (char const * *str*)

deserializes the string representation of a geometry transformation into a handle useable by API functions.

Parameters

str	string representation of a transformation.

Returns

upon success, handle to a new geometry transformation.

4.4.1.5.11 TRPIAPI char const* TRPI_GetCaptureSourceConfiguration (TRPI_Handle *source*)

fetch complete source configuration

this function obtains the complete current configuration set from the source and returns it as an XML formatted string.

Parameters

source	handle to capture source

Returns

XML formatted string

4.4.1.5.12 TRPIAPI char const* TRPI_GetCaptureSourceName (TRPI_Int *num*)

Returns the name of a capture source.

Parameters

num	numerical index of the capture source to be queried, ranging from 0 to N-1, where N is the return value of TRPI_getNumCaptureSources()

4.4.1.5.13 TRPIAPI TRPI_CaptureState TRPI_GetCaptureStatus (TRPI_Handle *source*)

obtains the operational status of a capture source

Parameters

source	Handle to capture source

Returns

status of the capture. Will be TRPI_CaptureNone if the source parameter is invalid.

4.4.1.5.14 TRPIAPI TRPI_Error TRPI_GetImageChannelNumber (TRPI_Handle *image*, TRPI_Int * *value*)

returns the number of channels in an image.

Parameters

image	Handle to image
value	pointer to variable that will receive the result

4.4.1.5.15 TRPIAPI TRPI_Error TRPI_GetImageDataPointer (TRPI_Handle *image*, void ** *pointer*)

returns pointer to raw image data

Parameters

image	Handle to image
pointer	Pointer to pointer, will be filled with the address of the image data.

4.4.1.5.16 TRPIAPI TRPI_Error TRPI_GetImageHeight (TRPI_Handle *image*, TRPI_Int * *value*)

returns the height of an image in pixels

Parameters

image	Handle to image
value	Pointer to an integer which will receive the value

4.4.1.5.17 TRPIAPI TRPI_Error TRPI_GetImageResolutionX (TRPI_Handle *image*, TRPI_Float * *value*)

returns the resolution of an image in X (scan line) direction

Parameters

image	Handle to image
value	Pointer to a variable which will receive the result.

4.4.1.5.18 TRPIAPI TRPI_Error TRPI_GetImageResolutionY (TRPI_Handle *image*, TRPI_Float * *value*)

returns the resolution of an image in Y (transport) direction

Parameters

image	Handle to image
value	Pointer to a variable which will receive the result.

4.4.1.5.19 TRPIAPI TRPI_Error TRPI_GetImageSampleFormat (TRPI_Handle *image*, TRPI_SampleFormat * *value*)

returns the sample of an image in pixels

Parameters

image	Handle to image
value	pointer to variable that will receive the result

4.4.1.5.20 TRPIAPI TRPI_Error TRPI_GetImageStride (TRPI_Handle *image*, TRPI_Int * *value*)

returns the stride of an image in Bytes (the number of bytes between two adjacent scan lines; due to padding, this may sometimes differ from the value computed from sample format, image dimension and number of channels)

Parameters

image	Handle to image
value	Pointer to an integer which will receive the value

4.4.1.5.21 TRPIAPI TRPI_Error TRPI_GetImageType (TRPI_Handle *image*, TRPI_ImageType * *value*)

returns the type of an image

Parameters

image	Handle to image
value	Pointer to variable that will receive the result

4.4.1.5.22 TRPIAPI TRPI_Error TRPI_GetImageWidth (TRPI_Handle *image*, TRPI_Int * *value*)

returns the width of an image in pixels

Parameters

image	Handle to image
value	Pointer to an integer which will receive the value

4.4.1.5.23 TRPIAPI TRPI_Error TRPI_GetLastError ( )

Returns the last error that occured in an API call.

For functions that have no explicit error return, call this function to obtain the code of the error that occured in the last call.

Returns

error value

4.4.1.5.24 TRPIAPI TRPI_Int TRPI_GetNumCaptureSources ( )

Queries for the number of currently supported capture sources.

Returns

the number of currently available capture sources

4.4.1.5.25 TRPIAPI TRPI_Handle TRPI_GetSourceWorkImage (TRPI_Handle *source*)

returns the work image associated with a capture source

each capture source has a work image memory associated with it, in which a captured image is stored. Use this function to obtain a handle to the work image. You can obtain a pointer to the image data, but the data should be treated as constant. In any case, do not store this handle persistently: the properties of the work image may change, the image data pointer may be freed or moved or the handle may become invalid altogether. Always copy work image data or use the job descriptor mechanism.

Parameters

source	Source handle.

4.4.1.5.26 TRPIAPI TRPI_Error TRPI_GetTransformNetImageSize (TRPI_Handle *transformation*, TRPI_Int * *maxWidth*, TRPI_Int * *maxHeight*)

computes the maximum net image size available after a geometry transformation if the "autocrop" configuration parameter is set to "1".

geometry transformation results in a maximum net output image available for requests. This function returns this size in pixels.

Parameters

transformation	handle to a previously created or restored transformation.
maxWidth	pointer to an integer to receive the maximum image width available after transformation, or NULL if you're not interested in the value.
maxHeight	pointer to an integer to receive the maximum image height available after transformation, or NULL if you're not interested in the value.

4.4.1.5.27 TRPIAPI TRPI_Error TRPI_Initialize ( )

Initializes the API.

Returns

TRPI_Ok or error code

4.4.1.5.28 TRPIAPI TRPI_Error TRPI_InitWhiteRefCache (TRPI_Handle *image*, TRPI_Int *use_configuration*, TRPI_Int *first_row*, TRPI_Int *last_row*)

Initializes the white reference cache.

This function initializes the white reference cache required for performing white referencing using stored data rather than computing it from every frame.

The white reference cache is used only when the global parameter "WR_mode" is set to "cached" and the parameters "WR1" to "WR4" are set!. See function TRPI_SetConfigurationString on how to change these parameters.

Parameters

handle	to an image that is used to fill the white reference cache. This can be either a fresh scan or an image stored to and reloaded from disk for this purpose.
use_configuration	when nonzero, the image is assumed to contain a white reference area as it is used for computing the white reference in each frame (in particular, the parameters "whiterefstart" and "whiterefend" are set correctly). If this value is NULL, the configuration is ignored and the values first_row and last_row are used instead.
first_row	if use_configuration is zero, this value must contain the first row, *in pixels*, of the area in the image that can be used for computing the white reference.
first_row	if use_configuration is zero, this value must contain the last row, *in pixels*, of the area in the image that can be used for computing the white reference. Set this parameter to a very large value, say 1e6, to use the entire image.

the following figure illustrates the meaning of first_row and last_row.

Figure 4.1: white reference cache

See Also

TRPI_SetConfigurationString

4.4.1.5.29 TRPIAPI TRPI_Int TRPI_JobDescriptorAddRequest (TRPI_Handle *descriptor*, TRPI_ImageType *type*, TRPI_Int *roiStartX*, TRPI_Int *roiStartY*, TRPI_Int *roiEndX*, TRPI_Int *roiEndY*)

adds a new processing request to a job descriptor.

A job descriptor is a list of ROI's and target types to be captured from a source or computed from an existing raw image.

All ROI coordinates must fit within the maximum net result size.

Parameters

descriptor	handle to a job descriptor
type	a supported image type.
roiStartX	leftmost column of the ROI
roiStartY	topmost row of the ROI
roiEndX	rightmost column of the ROI
roiEndY	bottommost row of the ROI (all in pixels)

Returns

result ID of the request; can later be passed to TRPI_jobDescriptorGetResult()

4.4.1.5.30 TRPIAPI TRPI_Int TRPI_JobDescriptorAddRequestEx (TRPI_Handle *descriptor*, TRPI_ImageType *type*, TRPI_SampleFormat *format*, TRPI_Int *roiStartX*, TRPI_Int *roiStartY*, TRPI_Int *roiEndX*, TRPI_Int *roiEndY*)

adds a new processing request to a job descriptor.

A job descriptor is a list of ROI's and target types to be captured from a source or computed from an existing raw image.

All ROI coordinates must fit within the maximum net result size.

Parameters

descriptor	handle to a job descriptor
type	a supported image type.
format	format for output image. Currently supported formats: TRPI_Format8U and TRPI_FormatFloat. This parameter is ignored for output types other than SRGB and LAB.

roiStartX	leftmost column of the ROI
roiStartY	topmost row of the ROI
roiEndX	rightmost column of the ROI
roiEndY	bottommost row of the ROI (all in pixels)

Returns

result ID of the request; can later be passed to TRPI_jobDescriptorGetResult()

4.4.1.5.31 TRPIAPI TRPI_Handle TRPI_JobDescriptorGetResult (TRPI_Handle *descriptor*, TRPI_Int *id*)

returns a request image

This function returns a handle to the request image identified by id.

Parameters

descriptor	handle to a job descriptor
id	an ID as returned by TRPI_jobDescriptorAddRequest

4.4.1.5.32 TRPIAPI void* TRPI_JobDescriptorGetResultAverage (TRPI_Handle *descriptor*, TRPI_Int *id*, TRPI_Int * *numValues*)

returns average over a request image region.

This function returns the average over all pixels of a request region.

Parameters

descriptor	handle to a job descriptor
id	an ID as returned by TRPI_jobDescriptorAddRequest
numValues	receives number of components in result.

Returns

pointer to result

4.4.1.5.33 TRPIAPI TRPI_Handle TRPI_LoadImage (char const * *file_name*)

loads an image from a TIF file

Parameters

file_name	string containing the name of a file. The name is automatically complemented with .TIF if it has no extension.

4.4.1.5.34 TRPIAPI TRPI_MultiMeasurementResult const* TRPI_MeasureLabeledComponents (TRPI_Handle *LCI*, TRPI_Handle *MC*, int * *numResults*)

Measurement on labeled segment image.

Each pixel in the given RGB image must have a value equal to the ID of the labeled segment it belongs to. Zero means that the pixel is background. Only the lower 16 bits (channels R and G) are evaluated, allowing for a total of 2^16-1 labeled segments.

Parameters

LCI	Handle to labeled component image
MC	Handle to multichannel image
numResults	receives the number of components (segments) for which a result was computed.

4.4.1.5.35 TRPIAPI TRPI_MultiMeasurementResult const* TRPI_MultiMeasurementResultEx (TRPI_Int *LCIwidth*, TRPI_Int *LCIheight*, TRPI_Int *LCIstride*, void * *LCIdata_pointer*, TRPI_Handle *MC*, int * *numResults*)

Measurement on labeled segment image in user memory.

This function behaves just like **TRPI_MultiMeasurementResult** (p. 12), exept for the fact that the labeled segment image is not passed as a handle, but instead directly by its attribute and data pointer.

The image defined by LCI{...} *must* be 24 bits per pixel RGB, hence the number of channels is assumed to be 3 and the sample format to be TRPI_TypeU8.

Parameters

LCIwidth	LCI image width in pixels
LCIheight	LCI image height in pixels
LCIstride	LCI image stride in bytes
LCIdata_pointer	pointer to first pixel in memory
MC	handle to multichannel image
numResults	receives the number of components (segments) for which a result was computed.

4.4.1.5.36 TRPIAPI void* TRPI_QueryCaptureSourceProperty (TRPI_Handle *source*, char const * *value_name*, TRPI_Error * *result*)

queries a single configuration value from a source

Parameters

source	handle to capture source
value_name	name part of the property/value pari
result	if non-NULL, receives a result error value

Returns

The type of the return value is determined by the type of the property; it's up to the caller to cast this to the appropriate type and use it accordingly.

4.4.1.5.37 TRPIAPI TRPI_Error TRPI_SaveImage (TRPI_Handle *image*, char const * *file_name*)

saves an API image to disk in the TIFF image format

Parameters

image	handle to image
file_name	string containing the name of a file. The name is automatically complemented with .TIF if it has no extension.

4.4.1.5.38 TRPIAPI char const* TRPI_SerializeTransform (TRPI_Handle *transformation*)

serializes the opaque transformation handle into a string representation for storage

Parameters

transformation	Handle to a valid transformation

Returns

pointer to a string representation of the transformation. The string is not guaranteed to be persistent and should be copied immediately by the caller.

4.4.1.5.39 TRPIAPI TRPI_Error TRPI_SetCaptureCallback (TRPI_Handle *source*, TRPI_ProgressCallbackFunction *callback*)

sets progress callback function for capture source

Associates a callback function with the source. Depending on the type of source and its configuration, the callback function may be called periodically or just once upon completion or error (with values 0.0 or 1.0, respectively). You should always call TRPI_getCaptureState to obtain further information.

Parameters

source	Handle to capture source
callback	pointer to a progress callback function.

4.4.1.5.40 TRPIAPI TRPI_Error TRPI_SetCaptureSourceProperty (TRPI_Handle *source*, char const * *value_name*, void * *new_value*)

sets a single configuration value in a source

Parameters

source	Handle to capture source
value_name	name of configuration value
new_value	new value for the property. It's up to the caller to supply the correct type.

4.4.1.5.41 TRPIAPI TRPI_Error TRPI_StopCapture (TRPI_Handle *source*)

stops a running image capture

Parameters

source	Handle to capture source

4.4.1.5.42 TRPIAPI TRPI_Error TRPI_TransformImage (TRPI_Handle *raw_image*, TRPI_Handle *transformation*, TRPI_Handle *job_descriptor*)

produces a job result from a raw image

This function takes a raw scan as input and, given a translation and a job descriptor, computes the requested output images.

Parameters

raw_image	handle to an image that was either scanned from a capture source, loaded from disk or otherwise supplied by the user in a format required by the API. The image must have dimensions compatible with the transformation and have the image type TRPI_typeRGB (user or unspecified RGB).
transformation	handle to a valid transformation (restored or created) compatible with raw_image.
job_descriptor	handle to a job descriptor listing the requested output images.

4.4.1.5.43 TRPIAPI TRPI_Handle TRPI_TransformToFastRGB (TRPI_Handle *transformation*, TRPI_Handle *raw_image*, TRPI_Handle *result_image*)

computes an overview RGB from a raw image

This function can be used to compute an overview RGB image, useable for preview or basic processing, from the raw TruePIXA input image. The computation is very fast, but does not produce accurate sRGB data - it outputs standard camera RGB instead.

The function operates just like TRPI_TransformImage, but it only returns a full TruePIXA image (no ROI) with 24 bits/pixel sample depth.

Parameters

transformation	Handle to an existing transformation
raw_image	the raw input image
result_image	handle to an image that will receive the output. This way, an existing image can be used without creating a new object. The image must have matching size and sample format. If this parameters is zero, the function creates a new image object with matching properties that may be re-used in the next call to this function.

Returns

handle to an image object - identical to result_image if the result data was written to that image, a new handle if no image was passed in, or TRPI_InvalidHandle in the case of an error. Be sure to check the return value of **TRPI_GetLastError()** (p. 21);

4.4.1.5.44 TRPIAPI char const* TRPI_TranslateError (TRPI_Error *error*)

maps an error code to an error text

Parameters

error	error code

Returns

string containing the error description or ("unknown error") if none was found

5

API Parameters

This section lists all parameters known to the TruePIXA API. They can be specified either by modifying the central configuration file located in `%ProgramData%/Chromasens/TpApi/tpapi.xml` or by using the API functions `TRPI_SetConfigurationString`, `TRPI_SetConfigurationInt` and `TRPI_SetConfigurationFloat`. Likewise, the current value of each parameter can be queried by the corresponding `TRPI_GetConfiguration...()` functions.

Parameter name	Type	Default value	Description
do_camctl	B	0	If 1 (true), the API attempts to initialize a video connection to the camera, enabling the API capture source for Matrix Vision frame grabbers. Note that setting this to 1 blocks the capture device for as long as the API DLL is loaded. Setting this flag to 1 requires the presence of the MatrixVision software stack.
clindex	I	0	Index of the CameraLink port to which the API will connect.
sync_threshold	I	128	When using software-based image trigger, this value indicates the video level which will trigger image capture.
camconffile	S	"%ProgramData%\ chromasens\ TpApi\ LineScanBuffered.xml"	Tells the API where to find the configuration file (setting XML) for the frame grabber
meascondition	S	"M1"	Names the currently active measurement condition. The "measurement condition" affects some internal parameters and settings during image capture, image processing and spectral reconstruction.
camoutputmode	S	"8bit/ch"	Tells the API how to treat image data received via TRPI_LoadImage, TRPI_CreateNewImage or from a capture source. This flag is highly specific to a particular TruePixa hardware revision and should not be modified without expert knowledge. other legal values: "16bit/ch (odd)", "16bit/ch (even)", "16bit/ch (avg)"
dynamicshading	S	"after geo. transform"	Defines if and when to perform dynamic white reference. When set to "after geo. transform", white reference is performed on the multichannel image after transformation. When set to "before geo. transform", WR is performed on the raw image (but *after* resolution correction)

Parameter name	Type	Default value	Description
dynamicshadingminlevel	F	15.0	When performing dynamic shading, pixels with video values below this level are not subject to shading correction.
dynamicshadingtargetlevel	F	240.0	Shading correction modifies the video values such that in the reference area, the average video level will be equal to this parameter.
autocrop	B	0	When 1, the API will determine the intersection of visible pixels in all four sub images and crop the output image to that range. Use this flag with caution! When doing spectral reconstruction, it is vitally important that color calibration data be in match with the resulting output image width. We currently recommend to leave this flag zero at all times and rather crop the result images.
horizontalflip	B	0	When 1, the output images will be mirrored in horizontal direction by the API. Do not modify this flag arbitrarily! This value must match the setting that was used for color calibration.
verticalflip	B	0	When 1, the output images will be mirrored vertically.
input_sequence	S	"RGB"	When creating multichannel images, assume the sequence of channels in the raw input image to be either "RGB" or "auto" to use the information from the image itself (assuming it to be correct)
defaultresx	F	300.0	Images loaded with TRPI_LoadImage() will be assigned this resolution in X direction if there is no resolution tag in the TIFF file.
defaultresy	F	300.0	Images loaded with TRPI_LoadImage() will be assigned this resolution in Y direction if there is no resolution tag in the TIFF file.
override_resolution	B	0	If set to 1, the resolution values given by defaultresx and defaultresy will be used in any case, whether or not there is resolution information in the TIF file.
whiterefstart	F	-1e10	Indicates the beginning (first image row) of the white reference are in the image. The meaning of this parameter depends on the setting of "use_syncmarker": if use_syncmarker is set to 1, then whiterefstart measures the *relative* distance from the center of the sync marker to the beginning of the white reference area. With use_syncmarker set to 0, the measure is relative to scan line 0. *Please note that the unit of this parameter is millimeter!* For the API to compute the correct position, the distances must translate into the proper pixel coordinates under the resolution specified either in the image loaded from disk or during TRPI_CreateNewImage(...). If the resolution so given is asymmetrical, i.e. $res_x \neq res_y$, the distances must be correct under res_x, since the API will reduce the image in transport direction such that $res_x = res_y$ by scaling the raw input by $f = \frac{res_y}{res_x}$ in transport direction. For this reason it is required that res_y always be larger than or equal to res_x.
whiterefend	F	-1e10	Marks the last row of the white reference area in a frame; see "whiterefstart"

Parameter name	Type	Default value	Description
WR_mode	S	"configuration"	Selects the mode of the white reference function. When set to "configuration", white reference works as detailed in the description of parameter "whiterefstart". When set to "cached", white reference is not computed from the image during TRPI_TransformImage; the white reference cache memory is used instead. This cache can be filled using the API function TRPI_InitWhiteRefChache; see the reference manual for details.
WR_default_file	S	empty string	Using this parameter, a default white reference image can be specified. The image is loaded upon API initialization if it exists.
showWRarea	B	0	Set this parameter to 1 to have the API highlight the white reference area in the image; can be used for troubleshooting purposes.
use_syncmarker	B	0	Set this parameter to 1 to activate the sync marker functionality. When enabled, the API will search for the synchronisation marker and compute the WR position relative to its center. When 0, WR position is relative to the image start.
rawmarkerfrom	I	-1	Indicates the column *in pixels* in the raw input image at which the search range for the sync marker begins. The marker will be searched over the entire height of the image and in the horizontal range defined by rawmarkerfrom and rawmarkerto.
rawmarkerto	I	-1	Indicates the column *in pixels* in the raw input image at which the search range for the sync marker ends.
rawmarkerchannel	I	1	Sets the color channel in the raw input image in which the sync marker is to be detected. Using the green channel is recommended.
vertswbinning	S	"average"	When reducing the image in transport direction as described in "whiterefstart", by default the reduction is done by averaging $\left\lfloor \frac{res_y}{res_x} \right\rfloor$ rows into one output row and then reducing the resulting image in Y direction by $f_y = \frac{\frac{res_y}{res_x}}{\left\lfloor \frac{res_y}{res_x} \right\rfloor}$ Set this value to "subsample" to copy only every $\left\lfloor \frac{res_y}{res_x} \right\rfloor$-th scanline to the intermediate image before correcting the remaining resolution imbalance by scaling.
darkcorrection	B	0	Activates dark offset correction in the API.
ctfilter	B	0	Activates CT correction in the API.

Key: B Boolean value ("0" or "1"), I integer value $-2^{31} \ldots 2^{31} - 1$, F real value (floating point), S: string value.