Software Developpement Kit
for

SmartIR 640

version 2.0.11

Provided by

Device-ALab

Monday 24th June, 2019, 10:38

Contents

Chapter 1

SmartIR640 SDK.

1.1 Introduction

Device-ALab provides a set of functions in a Library to communicate with *SmartIR640*.

This Library allows customers to use it's owns programming language (*C* ...) or tools (*Labview*, *Matlab* ...).

Provided SmartViewer GUI is based on features exposed by this Library.

1.1.1 Windows

Library kit provides the following items :

- DALProxy640USB.dll
- DALProxy640USB_x64.dll
- DALProxy640USB.h
- DALProxy640USBDef.h

These files must remains together.

Requirement

Microsoft Visual Studio 2010 Runtime is also required. This runtime is installed during *SmartIRViewer* setup.

1.1.2 Linux :

Library kit provides the following items :

- DALProxy640USB.so.x.y.z (shared library)
- DALProxy640USB.so.x.y (symbolic link)
- DALProxy640USB.so.x (symbolic link)
- DALProxy640USB.so (symbolic link)
- DALProxy640USB.h
- DALProxy640USBDef.h

(x.y.z is the version number)

Requirement

At least gcc 4.7.

1.2 Installation

DALProxy640USB Library was designed for Microsoft Windows Vista/Seven/8.X/10 and Linux.

SmartIR640 must be properly installed on system, plugged and powered.

1.2.1 Windows Installation :

In order to use the SmartIR640 module : install drivers. Use the library files to build programs you can share with msi drivers installer.

1.2.2 Linux Installation :

Don't forget to put the .so files into correct environment folder, or configure environment to point to the .so files.

1.3 Overview

Before using this Library, please take few minutes to read these Technical Notes .

Next, have a look at How to integrate this Library for details on how to use this Library in your favorite tools.

Then, the SDK came with some examples code, you can give a look at them to show how simple is this SDK.

Mode details about Library processing chain may be found on Processing Chain.

Calibration is a important part of IR image processing. To understand how to use SmartIR640 Calibration functions, please, give a look at Calibration Process.

1.4 Library Overview

Library provides a sub-set of functions :

- SmartIR640 Management
- SmartIR640 Processing
- SmartIR640 Image
- SmartIR640 Storage
- SmartIR640 Calibration
- Function return code

Chapter 2

Library Revision - History

2.1 Current version : version 2.0.11

- Adjust IrLugX320 to new ATTO320-60HZ LWIR detector
- Remove patch of last line for IrLugX320

2.2 Current version : version 2.0.10

- Handle VID/PID for new module version, fixing BOS descriptor compatibility.
- Fix shutter-less calibration with bad pixel cluster.
- Fix shutter-less save reliability.
- Fix cancelling calibration does not restore AGC state.

2.3 version 2.0.9

- Add IrLugX640 support.
- Fix IrLugX320 last line.
- Change IrLugX320 SDK API.

2.4 version 2.0.8

- Add Shutterless version 1.1.
- Upgrade Shutterless 1.0 to Shutterless 1.1.
- Remove support for SmartIR320 device.
- Add support for IRLugX320 device.

2.5 version 2.0.7

- Fix the GUI of Built In Self Test.

2.6 version 2.0.6

- Add the interface to save and restore the shuterless settings.

2.7 version 2.0.5

- Add support for Build-In Self Test.
- add support for SmartIR384C device.

2.8 version 2.0.4

- Add support for native x64 instruction set under Windows 7/8.1/10 64 bits
- Add support for Linux x86 and x64

2.9 Initial version : version 2.0.3

- Initial SmartIR SDK Family Release.
- Support for x86 instruction set under Windows 7/8.1/10.
- Support for Windows 7/8.1/10 64 bits with backward compatibility with x86 instruction set.
- Support for Linux Armv7 (soft-float) instruction set.
- Support for Linux Armv7 (hard-float) instruction set.
- Support for Linux Armv8 (hard-float) instruction set.

Chapter 3

Technical Notes

Important notes on using these functions.

3.1 Module connection

Several *SmartIR640* may be plugged into Workstation.

Due to image flow design, **only one application can be connected to *SmartIR640***. Application must release it (disconnect) to make it available to another application.

On another side, a single application can connect to several *SmartIR640*, and get images from them.

Note

> Even if only one module may be connected to one application, functions provide by DALProxy640USB Library can be used in a thread. For example, get image from a single module can be called from a separate thread, having one thread per *SmartIR640*. In the meanwhile, main thread (usually GUI) can call *Settings* functions for parameters update.

3.2 Data

DALProxy640USB Library was designed to be easily use by any programming langage or software able to use Library. Functions perform single task, and did not require special knowledge.

Functions provide by this Library use common C type :

- signed or unsigned char (1 byte).
- signed or unsigned short (2 bytes).
- signed or unsigned int (4 bytes).
- float (4 bytes).
- C Style string (null terminate array).

Note

> Most functions use HANDLE type. This type is void$*$.

3.3 Memory management

Note

In this section, caller refer as program calling Library function.

Library was designed to exchange many data with caller.

To simplify memory management, Library involves this single rule : **It's caller responsability to handle parameter placeholder**.

For example, when caller want to set a new table of Gain for NUC processing, caller allocates Gain table for values, and fills it. Then, it calls appropriate function, and passes pointer on this table as function parameter.

The same schema apply when caller want to retrieve table of Gain from NUC processing. Caller allocates Gain table for values, and passes it as parameter to appropriate function.

Warning

Most function use pre-defined table size (Gain, Offset and Image). It's caller responsibility to ensure table is large enough. Otherwise, memory corruption or even crash may occur.

Most table are the same number of element, i.e. image's dimension (`640 x 480 = 300K` values). But, depending on single value memory size (short vs float), memory allocation may be different.

On a final note about memory management : Caller don't need to hold memory allocation after calling Library fonction. Library fonction don't take ownership of parameters.

3.4 Dimension

This last point remind array dimension use by the Library. Image is 640 width by 480 height. Image data storage is `640 x 480 = 300KPixels = 600KB` values. The same dimension apply to :

- Gain values table.
- Offset values table.

Bad pixel table is limited to 1023 elements. Place holder for bad pixels retrieval must be large enough.

SmartIR640 provides 8 slots for Gain or Offset table.

Chapter 4

Processing Chain

Which steps are performed before the delivery of images.

SmartIR640 include an Image Processing Chain.

For more information about *processing chain*, see *User's Guide - Image processing*.

This processing chain is composed of the following steps :

- Non Uniformity Correction.
- Bad Pixel Correction.
- Automatic Gain Correction

These 3 steps are disabled by default. Each of them may be disabled using Proxy640USB_SetProcessing() function.

4.1 Non Uniformity Correction

NUC require Gain and Offset values for each image pixel. Default Gain and Offset at Library startup provide a neutral NUC, i.e. do not modified raw image. Gain values may be set using Proxy640USB_SetTableGain(), and Offset values my be set using Proxy640USB_SetTableOffset().

Current Gain and Offset table may be query using Proxy640USB_GetTableGain() and Proxy640USB_GetTable-Offset().

SmartIR640 can store Gain and Offset table into slot. See SmartIR640 Storage.

User can provide his own values, or use calibration process. See SmartIR640 Calibration.

Note

See *User's Guide - Full Calibration* for more details about calibration.

4.2 Bad Pixel Correction

This processing fixes bad pixel from *SmartIR640*. According bad pixel's position, some may not be fixable. Proxy640USB_GetPixelMask() build an image mask of pixel status. See function documentation for details. It required a list of pixel position (x, y), inside image ([0-639],[0-479]). To set bad pixel list, use Proxy640USB_Set-BadPixels(). Proxy640USB_GetBadPixels() is used to retrieve current pixel list. *SmartIR640* can store pixel list using Proxy640USB_SaveBadPixels(), and retrieve it with Proxy640USB_LoadBadPixels().

Note

Bad pixel list is limited to 1023 pixels.

4.3 Automatic Gain Correction

Images can be processed with gain correction, with different automatic method. Its purpose is to maximize the image dynamic. To configure Automatic Gain Correction, use Proxy640USB_SetAGCProcessing. The possibility are Disable (eNoAGC), Histogram (eAGCEqHisto), Enhanced rendering (eAGCEnhanced), Linear (eAGCLinear). Proxy640USB_GetAGCProcessing is used to retrieve current AGC processing.

Note

see *User's Guide - Gain control* for more details about AGC.

Chapter 5

How to integrate this Library

Communicate with *SmartIR640* using your favorite tool.

DALProxy640USB Library may be used with a programming langage (like *C*), or tool (like *Labview*, *Matlab* ...).

Note

Functions provide in Library are C standard style.

5.1 Tools.

Tools like *Labview* from National Instruments allows library function call. Use `Call library function` node, and configure it to match function prototype using this documentation.

See Also

`National Instruments online help for Call library function node`.

Note

This DLL was designed for 32bit *Labview* version only.

Matlab from *Mathwork* use shared library in a way similar to C language. First, you have to load library using `loadlibrary()`, and call a function using `calllib()`. *Matlab* will require functions header definition.

See Also

`Mathwork online help for C Shared Library functions`.

Chapter 6

Calibration Process

How to perform calibration for *SmartIR640*.

The SDK provides functions for :

- Calibration for Shutter mode :
- Shutterless Calibrations :

Note

See *User's Guide* - *Calibration for Shutter mode* and *Calibration for Shutter less mode* for more details.

6.1 Calibration for Shutter mode :

6.1.1 Full Calibration

This calibration, upon successfull with generate :

- Gain table,
- Offset table,
- Bad pixel list.

Warning

Upon completion, Gain, Offset and Bad pixel data currently in use will be overwritten.

To make this calibration, the correct sequencing is : In front of low temperature black body :

- Proxy640USB_InitShutter2PtsCalibration for stage 1
- Call several times Proxy640USB_StepShutter2PtsCalibration for stage 1 in order to capture low temperature images
- Proxy640USB_FinishShutter2PtsCalibration for stage 1

In front of high temperature black body :

- Proxy640USB_InitShutter2PtsCalibration for stage 2
- Call several times Proxy640USB_StepShutter2PtsCalibration for stage 2 in order to capture high temperature images

- Proxy640USB_FinishShutter2PtsCalibration for stage 2

At the end of the process, on success new calibration data is set.

The Proxy640USB_StepShutter2PtsCalibration capture the current sensor image and must be called several times to reduce temporal noise.

Warning

During the Calibration sequence, all processing are disabled and must not be enable again during calibration. At the end of the process, the processing setting are restored back.

6.1.2 Fast Calibration

This calibration, also known as *Shutter Calibration*, or one point calibration will only produce new Offset values.

Note

Current Gain and Bad pipxel list will remain untouched.

To make this calibration, the correct sequencing is :

In front of black body or shutter:

- Proxy640USB_InitShutterCalibration
- Call several times Proxy640USB_StepShutterCalibration in order to capture images
- Proxy640USB_FinishShutterCalibration

At the end of the process, on success new Offset are calculate and set, using current Gain values and Bad Pixels.

The Proxy640USB_StepShutterCalibration add the current sensor image and must be called several times to reduce temporal noise.

Warning

During the Calibration sequence, NUC processing is disabled and must not be enable again during calibration. At the end of the process, the processing setting are restored back.

6.2 Shutterless Calibrations :

This calibration is based on existing bad pixel list.

6.2.1 Shutter less Calibration T0

This calibration, upon successful will generate :

- Shutter less data for FPA temperature T0.

Warning

Upon completion, shutter less data currently in use will be overwritten.

To make this calibration, the correct sequencing is :

In front of low temperature black body :

- Proxy640USB_InitSLCalibrationT0 for stage 1
- Call several times Proxy640USB_StepSLCalibrationT0 for stage 1 in order to capture low temperature images
- Proxy640USB_FinishSLCalibrationT0 for stage 1

In front of high temperature black body :

- Proxy640USB_InitSLCalibrationT0 for stage 2
- Call several times Proxy640USB_StepSLCalibrationT0 for stage 2 in order to capture high temperature images
- Proxy640USB_FinishSLCalibrationT0 for stage 2

At the end of the process, on success new calibration data is set.

The Proxy640USB_StepSLCalibrationT0 capture the current sensor image and must be called several times to reduce temporal noise.

Warning

During the Calibration sequence, all processing are disabled and must not be enable again during calibration. At the end of the process, the processing setting are restored back.

6.2.2 Shutter less Calibration T1

This calibration, upon successful will generate :

- Shutter less data for FPA temperature T1.

To make this calibration, the correct sequencing is :

In front of black body or shutter :

- Proxy640USB_InitSLCalibrationT1
- Call several times Proxy640USB_StepSLCalibrationT1 in order to capture temperature images
- Proxy640USB_FinishSLCalibrationT1

At the end of the process, on success new calibration data is set.

The Proxy640USB_StepSLCalibrationT1 capture the current sensor image and must be called several times to reduce temporal noise.

Warning

During the Calibration sequence, all processing are disabled and must not be enable again during calibration. At the end of the process, the processing setting are restored back.

Chapter 7

Module Index

7.1 Modules

Here is a list of all modules:

Chapter 8

Module Documentation

8.1 SmartIR640 Management

Etablish and manage communication with SmartIR640.

Functions

- eDALProxy640USBErr Proxy640USB_GetModuleCount (int ∗paiCount)
- eDALProxy640USBErr Proxy640USB_GetModuleName (int iIdx, char ∗paName, int iLen)
- eDALProxy640USBErr Proxy640USB_ConnectToModule (int iIdx, HANDLE ∗paHandle)
- eDALProxy640USBErr Proxy640USB_IsConnectToModule (HANDLE paHandle)
- eDALProxy640USBErr Proxy640USB_DisconnectFromModule (HANDLE paHandle)
- eDALProxy640USBErr Proxy640USB_RunBIST (HANDLE paHandle, unsigned int ∗diagCode)

8.1.1 Detailed Description

Etablish and manage communication with SmartIR640. This set provides :

- Functions to enumerates and name plugged SmartIR640.
- Function to connect and disconnect to SmartIR640.

Application call Proxy640USB_GetModuleCount() to know how many SmartIR640 are plugged to workstation. First SmartIR640 index is 0, and so on.

Calling Proxy640USB_GetModuleCount() check SmartIR640 count. So, call it will refresh SmartIR640 list.

Before calling any other function's group, Application must connect to a SmartIR640 using Proxy640USB_ConnectToModule().

Once a SmartIR640 is connected by an application, it's not available to another application. Application must release SmartIR640 by calling Proxy640USB_DisconnectFromModule().

Connection to SmartIR640 will provide a *handle*. This *handle* is use by all functions addressing this SmartIR640. It remains valid until Proxy640USB_DisconnectFromModule() is called.

Application can connect several SmartIR640, using different *handles*.

8.1.2 Function Documentation

8.1.2.1 eDALProxy640USBErr Proxy640USB_GetModuleCount (int ∗ *paiCount*)

Retrieve current count of plugged module.

Parameters

out	*paiCount*	Number of plugged module.

8.1.2.2 eDALProxy640USBErr Proxy640USB_GetModuleName (int *iIdx,* char ∗ *paName,* int *iLen*)

Query SmartIR640 name by index.

Parameters

in	*iIdx*	Module index.
out	*paName*	SmartIR640 name from index.
in	*iLen*	paName storage size.

8.1.2.3 eDALProxy640USBErr Proxy640USB_ConnectToModule (int *iIdx,* HANDLE ∗ *paHandle*)

Connect to SmartIR640 by index.

This function will return a handle, which will be uses as SmartIR640 identifier.

Connection may failed if SmartIR640 is already connected by another application.

Parameters

in	*iIdx*	Module index. First SmartIR640 index is 0.
out	*paHandle*	SmartIR640 handle.

8.1.2.4 eDALProxy640USBErr Proxy640USB_IsConnectToModule (HANDLE *paHandle*)

Check if handle connection. This function will check if handle is still valid, and then check connection with SmartI-R640.

Parameters

in	*paHandle*	SmartIR640 handle.

Returns

eProxy640USBSuccess on success, or error code.

8.1.2.5 eDALProxy640USBErr Proxy640USB_DisconnectFromModule (HANDLE *paHandle*)

Disconnect to SmartIR640 by index. This function will release SmartIR640 connection.

Parameters

in	*paHandle*	SmartIR640 handle.

8.1.2.6 eDALProxy640USBErr Proxy640USB_RunBIST (HANDLE *paHandle,* unsigned int ∗ *diagCode*)

Run the SmartIR640 Built-In Self-Test. This function will check if handle is still valid, and then run the built-in self tests.

Parameters

`in`	*paHandle*	SmartIR640 handle.
`out`	*diagCode*	Diagnostic code provided by the SmartIR640. Value is 0 in case of success.

Returns

eProxy640USBSuccess on success, or error code.

8.2 SmartIR640 Processing

Control SmartIR640 image processing. Query module connected to workstation, Open and close link.

Functions

- eDALProxy640USBErr Proxy640USB_SetCalibrationConfig (HANDLE paHandle, int paParam)
- eDALProxy640USBErr Proxy640USB_SetNUCProcessing (HANDLE paHandle, unsigned char paBadPixels, unsigned char paNUC)
- eDALProxy640USBErr Proxy640USB_GetNUCProcessing (HANDLE paHandle, unsigned char ∗paBadPixels, unsigned char ∗paNUC)
- eDALProxy640USBErr Proxy640USB_SetShutterLessProcessing (HANDLE paHandle, bool bActivate)
- eDALProxy640USBErr Proxy640USB_GetShutterLessProcessing (HANDLE paHandle, bool ∗pbIsActive)
- eDALProxy640USBErr Proxy640USB_SetAGCProcessing (HANDLE paHandle, unsigned char paeAGCProcessing)
- eDALProxy640USBErr Proxy640USB_GetAGCProcessing (HANDLE paHandle, unsigned char ∗paeAGCProcessing)
- eDALProxy640USBErr Proxy640USB_SetCurrentTableGain (HANDLE paHandle, float ∗paTableGains)
- eDALProxy640USBErr Proxy640USB_SetCurrentTableOffset (HANDLE paHandle, signed short ∗paTableOffsets)
- eDALProxy640USBErr Proxy640USB_SetCurrentBadPixels (HANDLE paHandle, unsigned short ∗paTableX, unsigned short ∗paTableY, unsigned short paCount)
- eDALProxy640USBErr Proxy640USB_SetCurrentShutterless (HANDLE paHandle, unsigned int ∗paShutterless)
- eDALProxy640USBErr Proxy640USB_GetCurrentShutterlessSize (HANDLE paHandle, unsigned int ∗pSize)
- eDALProxy640USBErr Proxy640USB_GetCurrentShutterless (HANDLE paHandle, unsigned int ∗paShutterless)
- eDALProxy640USBErr Proxy640USB_GetCurrentTableGain (HANDLE paHandle, float ∗paTableGains)
- eDALProxy640USBErr Proxy640USB_GetCurrentTableOffset (HANDLE paHandle, signed short ∗paTableOffsets)
- eDALProxy640USBErr Proxy640USB_GetCurrentBadPixels (HANDLE paHandle, unsigned short ∗paTableX, unsigned short ∗paTableY, unsigned short ∗paCount)

8.2.1 Detailed Description

Control SmartIR640 image processing. Query module connected to workstation, Open and close link. This set of function provides control over image processing.

- Query and change processing step state (enable or disable).
- Query processing parameters.
- Set processing parameters.

Processing is compose of :

- Bad pixel correction.
- Non linearity correction.

See Also

User's Guide or Processing Chain for details

8.2.2 Function Documentation

8.2.2.1 eDALProxy640USBErr Proxy640USB_SetCalibrationConfig (HANDLE *paHandle*, int *paParam*)

Configure Internal Calibration.

Parameters

in	*paHandle*	SmartIR640 handle.
in	*paParam*	configuration to apply. • bit[0] Enable (1) or Disable (0) the automatic fast calibration associated with mechanical shutter • bit[1-31] Reserved.

8.2.2.2 eDALProxy640USBErr Proxy640USB_SetNUCProcessing (HANDLE *paHandle,* unsigned char *paBadPixels,* unsigned char *paNUC*)

Enable/Disable NUC processing steps. These are enabled by default at connection.

Parameters

in	*paHandle*	SmartIR640 handle.
in	*paBadPixels*	Enable(1)/Disable(0) bad pixels correction.
in	*paNUC*	Enable(1)/Disable(0) Non Uniformity Correction.

Returns

This return error eProxy640USBFeatureNotAvailable if Shutterless is activated.

8.2.2.3 eDALProxy640USBErr Proxy640USB_GetNUCProcessing (HANDLE *paHandle,* unsigned char ∗ *paBadPixels,* unsigned char ∗ *paNUC*)

Query NUC processing steps status.

Parameters

in	*paHandle*	SmartIR640 handle.
out	*paBadPixels*	bad pixels correction enable(1) or disable(0).
out	*paNUC*	Non Uniformity Correction enable(1) or disable(0).

Returns

This return error eProxy640USBFeatureNotAvailable if Shutterless is activated.

8.2.2.4 eDALProxy640USBErr Proxy640USB_SetShutterLessProcessing (HANDLE *paHandle,* bool *bActivate*)

Enable/Disable ShutterLess processing.

Parameters

in	*paHandle*	SmartIR640 handle.
in	*bActivate*	Enable(true)/Disable(false) Shutterless processing.

Returns

This function return eProxy640USBFeatureNotAvailable error if Shutterless is unavailable on this module.

8.2.2.5 eDALProxy640USBErr Proxy640USB_GetShutterLessProcessing (HANDLE *paHandle,* bool ∗ *pbIsActive*)

Query Shutterless processing status.

Parameters

`in`	*paHandle*	SmartIR640 handle.
`out`	*pbIsActive*	shutterless processing enable(true) or disable(false).

8.2.2.6 eDALProxy640USBErr Proxy640USB_SetAGCProcessing (HANDLE *paHandle,* unsigned char *paeAGCProcessing*)

Set Auto Gain Control processing step. By default, No AGC processing set.

Parameters

`in`	*paHandle*	SmartIR640 handle.
`in`	*paeAGC-Processing*	see eAGCProcessingValue for values.

8.2.2.7 eDALProxy640USBErr Proxy640USB_GetAGCProcessing (HANDLE *paHandle,* unsigned char ∗ *paeAGCProcessing*)

Query processing steps status.

Parameters

`in`	*paHandle*	SmartIR640 handle.
`out`	*paeAGC-Processing*	see eAGCProcessingValue for values.

8.2.2.8 eDALProxy640USBErr Proxy640USB_SetCurrentTableGain (HANDLE *paHandle,* float ∗ *paTableGains*)

Set Gains values for NUC processing.

Parameters

`in`	*paHandle*	SmartIR640 handle.
`in`	*paTableGains*	New Gains values for NUC processing.

Note

Each pixel must have a value. So paTableGains must contains 640 ∗ 640 float values (4 bytes float).

8.2.2.9 eDALProxy640USBErr Proxy640USB_SetCurrentTableOffset (HANDLE *paHandle,* signed short ∗ *paTableOffsets*)

Set Offset values for NUC processing.

Parameters

`in`	*paHandle*	SmartIR640 handle.
`in`	*paTableOffsets*	New offsets values for NUC processing.

Note

Each pixel must have a value. So paTableGains must contains 640 ∗ 480 values (2 bytes signed value).

8.2.2.10 eDALProxy640USBErr Proxy640USB_SetCurrentBadPixels (HANDLE *paHandle,* unsigned short ∗ *paTableX,* unsigned short ∗ *paTableY,* unsigned short *paCount*)

Set bad pixels position in image for bad pixels correction.

Parameters

`in`	*paHandle*	SmartIR640 handle.
`in`	*paTableX,pa-TableY*	Bad pixels position in image.
`in`	*paCount*	bad pixels count.

8.2.2.11 eDALProxy640USBErr Proxy640USB_SetCurrentShutterless (HANDLE *paHandle,* unsigned int ∗ *paShutterless*)

Set Shutterless data for restore purpose. Shutterless data must be considered as binary and must not be modified.

Parameters

`in`	*paHandle*	SmartIR640 handle.
`in`	*paShutterless*	values.

8.2.2.12 eDALProxy640USBErr Proxy640USB_GetCurrentShutterlessSize (HANDLE *paHandle,* unsigned int ∗ *pSize*)

Get Shutterless size of the data for backup and restore purpose.

Parameters

`in`	*paHandle*	SmartIR640 handle.
`out`	*pSize*	size of shutterless data in bytes.

8.2.2.13 eDALProxy640USBErr Proxy640USB_GetCurrentShutterless (HANDLE *paHandle,* unsigned int ∗ *paShutterless*)

Get Shutterless data for backup purpose. Shutterless data must be considered as binary and must not be modified.

Parameters

`in`	*paHandle*	SmartIR640 handle.
`in`	*paShutterless*	New Shutterless values.

8.2.2.14 eDALProxy640USBErr Proxy640USB_GetCurrentTableGain (HANDLE *paHandle,* float ∗ *paTableGains*)

Get Gains current values from NUC processing.

Parameters

`in`	*paHandle*	SmartIR640 handle.
`in`	*paTableGains*	New Gains values for NUC processing.

Note

Each pixel must have a value. So paTableGains must contains 640 $*$ 480 float values (4 bytes float).

8.2.2.15 eDALProxy640USBErr Proxy640USB_GetCurrentTableOffset (HANDLE *paHandle,* signed short ∗ *paTableOffsets*)

Get Offset current values from NUC processing.

Parameters

`in`	*paHandle*	SmartIR640 handle.
`in`	*paTableOffsets*	New offsets values for NUC processing.

Note

Each pixel must have a value. So paTableGains must contains 640 ∗ 480 values (2 bytes signed value).

8.2.2.16 eDALProxy640USBErr Proxy640USB_GetCurrentBadPixels (HANDLE *paHandle,* unsigned short ∗ *paTableX,* unsigned short ∗ *paTableY,* unsigned short ∗ *paCount*)

Get current bad pixels position in image from bad pixels correction.

Parameters

`in`	*paHandle*	SmartIR640 handle.
`in`	*paTableX,pa-TableY*	Bad pixels position in image.
	paCount	Initial bad pixels array size, on return, bad pixel count.

Note

paCount must be init with paTableX / paTableY placeholder size (to avoid overflow), and will be modified by function with current bad pixel count.

8.3 SmartIR640 Control

Set or Get module features. Refer to module user guide for details on feature, and SDK header file for paeFeature definition.

Functions

- eDALProxy640USBErr Proxy640USB_GetStringFeature (HANDLE paHandle, int paeFeature, char ∗paStr)
- eDALProxy640USBErr Proxy640USB_GetUIntFeature (HANDLE paHandle, int paeFeature, unsigned int ∗paUInt)
- eDALProxy640USBErr Proxy640USB_GetFloatFeature (HANDLE paHandle, int paeFeature, float ∗paFloat)
- eDALProxy640USBErr Proxy640USB_SetStringFeature (HANDLE paHandle, int paeFeature, const char ∗paStr)
- eDALProxy640USBErr Proxy640USB_SetUIntFeature (HANDLE paHandle, int paeFeature, unsigned int paUInt)
- eDALProxy640USBErr Proxy640USB_SetFloatFeature (HANDLE paHandle, int paeFeature, float paFloat)

8.3.1 Detailed Description

Set or Get module features. Refer to module user guide for details on feature, and SDK header file for paeFeature definition.

8.3.2 Function Documentation

8.3.2.1 eDALProxy640USBErr Proxy640USB_GetStringFeature (HANDLE *paHandle,* int *paeFeature,* char ∗ *paStr*)

Query string feature.

Parameters

`in`	*paHandle*	SmartIR640 handle.
`in`	*paeFeature*	Feature requested.
`out`	*paStr*	String from requested feature.

Warning

String Feature are 32 byte large, including null byte. Ensure paStr is large enougt.

8.3.2.2 eDALProxy640USBErr Proxy640USB_GetUIntFeature (HANDLE *paHandle,* int *paeFeature,* unsigned int ∗ *paUInt*)

Query integer feature.

Parameters

`in`	*paHandle*	SmartIR640 handle.
`in`	*paeFeature*	Feature requested.
`out`	*paUInt*	Integer value from requested feature.

8.3.2.3 eDALProxy640USBErr Proxy640USB_GetFloatFeature (HANDLE *paHandle,* int *paeFeature,* float ∗ *paFloat*)

Query float feature.

Parameters

`in`	*paHandle*	SmartIR640 handle.
`in`	*paeFeature*	Feature requested.
`out`	*paFloat*	Float value from requested feature.

8.3.2.4 eDALProxy640USBErr Proxy640USB_SetStringFeature (HANDLE *paHandle,* int *paeFeature,* const char $*$ *paStr*)

Set string feature.

Parameters

`in`	*paHandle*	SmartIR640 handle.
`in`	*paeFeature*	Feature written.
`in`	*paStr*	String for written feature.

Warning

String Feature are 32 byte large, including null byte. Ensure paStr is large enougt.

8.3.2.5 eDALProxy640USBErr Proxy640USB_SetUIntFeature (HANDLE *paHandle,* int *paeFeature,* unsigned int *paUInt*)

Set integer feature.

Parameters

`in`	*paHandle*	SmartIR640 handle.
`in`	*paeFeature*	Feature written.
`in`	*paUInt*	Integer value for written feature.

8.3.2.6 eDALProxy640USBErr Proxy640USB_SetFloatFeature (HANDLE *paHandle,* int *paeFeature,* float *paFloat*)

Query float feature.

Parameters

`in`	*paHandle*	SmartIR640 handle.
`in`	*paeFeature*	Feature written.
`in`	*paFloat*	Float value for written feature.

8.4 SmartIR640 Image

Query Image from SmartIR640.

Functions

- eDALProxy640USBErr Proxy640USB_GetImage (HANDLE paHandle, unsigned short ∗paImage, int ∗paMeta, int paiTimeout)

8.4.1 Detailed Description

Query Image from SmartIR640. This set provides a single function to query current SmartIR640 image. Calling it will block application until an image is available, or timeout occurs.

Application may provide image storage for new IR image. Image nature (Raw or Fixed) depend on processing settings (see SmartIR640 Processing).

IR image is **640 width by 480 heigth**. Pixel storage is unsigned short, with 16bit effective, LSB aligned.

Along IR Image, some meta data are provides.

8.4.2 Function Documentation

8.4.2.1 eDALProxy640USBErr Proxy640USB_GetImage (HANDLE *paHandle*, unsigned short ∗ *paImage*, int ∗ *paMeta*, int *paiTimeout*)

Query image from SmartIR640.

Parameters

`in`	*paHandle*	SmartIR640 handle.
`out`	*paImage*	Image placeholder for new image. Must be at least *640 x 480 x 2= 600KB*.
`out`	*paMeta*	Meta-Data placeholder. Must be at least 135 32bit values : • [0] fpa temperature in celsius (cast float to get it). • [1] period from previous image (in microsecond). • [2] frame counter (16bit effective). • [3-4] time from epoch, using 64bit (use 2 values). • [5-6] Reserved. • [7-134] Histogram.

in	*paiTimeout*	Operation timeout in millisecond.

8.5 SmartIR640 Storage

Store and retrieve processing settings into SmartIR640.

Functions

- eDALProxy640USBErr Proxy640USB_StartupDefault (HANDLE paHandle, unsigned char ∗paIdxGains, unsigned char ∗paIdxOffsets, unsigned char ∗paIdxBank)
- eDALProxy640USBErr Proxy640USB_SlotType (HANDLE paHandle, unsigned char paIndex, unsigned char ∗paeType, void ∗paData)
- eDALProxy640USBErr Proxy640USB_LoadTableGain (HANDLE paHandle, unsigned char paIndex, float ∗paTableGain, void ∗paData)
- eDALProxy640USBErr Proxy640USB_LoadTableOffset (HANDLE paHandle, unsigned char paIndex, short ∗paTableOffset, void ∗paData)
- eDALProxy640USBErr Proxy640USB_LoadBadPixels (HANDLE paHandle, unsigned short ∗paTableX, unsigned short ∗paTableY, unsigned short ∗paCount)
- eDALProxy640USBErr Proxy640USB_SaveTableGain (HANDLE paHandle, unsigned char paIndex, const float ∗paTableGain, void ∗paData)
- eDALProxy640USBErr Proxy640USB_SaveTableOffset (HANDLE paHandle, unsigned char paIndex, const short ∗paTableOffset, void ∗paData)
- eDALProxy640USBErr Proxy640USB_SaveBadPixels (HANDLE paHandle, const unsigned short ∗paTableX, const unsigned short ∗paTableY, unsigned short paCount)
- eDALProxy640USBErr Proxy640USB_LoadCurrentTableGain (HANDLE paHandle, unsigned char paIndex)
- eDALProxy640USBErr Proxy640USB_LoadCurrentTableOffset (HANDLE paHandle, unsigned char paIndex)
- eDALProxy640USBErr Proxy640USB_LoadCurrentBadPixels (HANDLE paHandle)
- eDALProxy640USBErr Proxy640USB_SaveCurrentTableGain (HANDLE paHandle, unsigned char paIndex, const void ∗paData)
- eDALProxy640USBErr Proxy640USB_SaveCurrentTableOffset (HANDLE paHandle, unsigned char paIndex, const void ∗paData)
- eDALProxy640USBErr Proxy640USB_SaveCurrentBadPixels (HANDLE paHandle)
- eDALProxy640USBErr Proxy640USB_SaveCurrentShutterlessTables (HANDLE paHandle)
- eDALProxy640USBErr Proxy640USB_LoadCurrentShutterlessTables (HANDLE paHandle)

8.5.1 Detailed Description

Store and retrieve processing settings into SmartIR640. SmartIR640 provides **8** slots to store Gain or Offset value. Slot are not dedicated to a kind of data.

Attention

> Storage space is limited into SmartIR640. Hence, data (Gain or Offset) are rounded to fit into slot. This may involve difference if your store data, read it, and compare to your initial values. For coherence, this data reduction is also apply when update NUC processing data (see SmartIR640 Processing).

Save functions provides a *MakeDefault* parameter. When set to 1 (enable), this will mark slot as default. When application connect to SmartIR640, Proxy640USB_ConnectToModule() function will look for default slot, and load into processing data from slot.

8.5.2 Function Documentation

8.5.2.1 eDALProxy640USBErr Proxy640USB_StartupDefault (HANDLE *paHandle,* unsigned char ∗ *paIdxGains,* unsigned char ∗ *paIdxOffsets,* unsigned char ∗ *paIdxBank*)

Default slot index for Gain values and Offset values, last setting's bank used.

Parameters

in	*paHandle*	SmartIR640 handle.
out	*paiIdxGains*	Gain slot index, of 255 if no default Gain slot index.
out	*paiIdxOffsets*	Offset slot index, of 255 if no default Offset slot index.
out	*paiIdxBank*	Settings bank index, of 255 if no default settings index.

Note

No need to call and use this function (already done at SmartIR640 connection)

8.5.2.2 eDALProxy640USBErr Proxy640USB_SlotType (HANDLE *paHandle,* unsigned char *paiIndex,* unsigned char ∗ *paeType,* void ∗ *paData*)

Query slot data type.

Parameters

in	*paHandle*	SmartIR640 handle.
in	*paiIndex*	Slot index to query.
out	*paeType*	Slot type.
out	*paData*	Table associate data. NULL, or 60 bytes placeholder. paData is additional data associated to Gain or Offset array, which can be used freely by application for instance to keep a trace of Gain or Offset table calibration conditions, either sensitivity, either focal plane array temperature. Slot type value are : • 0 :Empty slot. • 1 :Gain values. • 2 :Offset values.

8.5.2.3 eDALProxy640USBErr Proxy640USB_LoadTableGain (HANDLE *paHandle,* unsigned char *paiIndex,* float ∗ *paTableGain,* void ∗ *paData*)

Retrieve SmartIR640 slot data as Gain values.

This function may failed if slot is empty, or slot data are not Gain values.

Parameters

in	*paHandle*	SmartIR640 handle.
in	*paiIndex*	Slot index as data.
out	*paTableGain*	Gain values from SmartIR640 slot.
out	*paData*	Table associate data. NULL, or 60 bytes placeholder. paData is additional data associated to Gain or Offset array, which can be used freely by application for instance to keep a trace of Gain or Offset table calibration conditions, either sensitivity, either focal plane array temperature.

8.5.2.4 eDALProxy640USBErr Proxy640USB_LoadTableOffset (HANDLE *paHandle,* unsigned char *paiIndex,* short ∗ *paTableOffset,* void ∗ *paData*)

Retrieve SmartIR640 slot data as Offset values.

This function may failed if slot is empty, or slot data are not Offset values.

Parameters

`in`	*paHandle*	SmartIR640 handle.
`in`	*paiIndex*	Slot index as data.
`out`	*paTableOffset*	Offset values from SmartIR640 slot.
`out`	*paData*	Table associate data. NULL, or 60 bytes placeholder. paData is additional data associated to Gain or Offset array, which can be used freely by application for instance to keep a trace of Gain or Offset table calibration conditions, either sensitivity, either focal plane array temperature.

8.5.2.5 eDALProxy640USBErr Proxy640USB_LoadBadPixels (HANDLE *paHandle,* unsigned short $*$ *paTableX,* unsigned short $*$ *paTableY,* unsigned short $*$ *paCount*)

Retrieve bad pixel position in image from SmartIR640.

Parameters

`in`	*paHandle*	SmartIR640 handle.
`out`	*paTableX,pa-TableY*	Bad pixels position in image.
	paCount	Initial bad pixels array size, on return, bad pixel count.

Note

paCount must be init with paTableX / paTableY placeholder size (to avoid overflow), and will be modified by function with current bad pixel count.

8.5.2.6 eDALProxy640USBErr Proxy640USB_SaveTableGain (HANDLE *paHandle,* unsigned char *paiIndex,* const float $*$ *paTableGain,* void $*$ *paData*)

Save Gain values into SmartIR640 slot data.

Parameters

`in`	*paHandle*	SmartIR640 handle.
`in`	*paiIndex*	Slot index.
`in`	*paTableGain*	Gain values to store into SmartIR640 slot.
`in`	*paData*	Table associate data. NULL, or 60 bytes placeholder. paData is additional data associated to Gain or Offset array, which can be used freely by application for instance to keep a trace of Gain or Offset table calibration conditions, either sensitivity, either focal plane array temperature.

8.5.2.7 eDALProxy640USBErr Proxy640USB_SaveTableOffset (HANDLE *paHandle,* unsigned char *paiIndex,* const short $*$ *paTableOffset,* void $*$ *paData*)

Save Offset values into SmartIR640 slot data.

Parameters

`in`	*paHandle*	SmartIR640 handle.
`in`	*paiIndex*	Slot index.
`in`	*paTableOffset*	Offset values to store into SmartIR640 slot.

in	*paData*	Table associate data. NULL, or 60 bytes placeholder. paData is additional data associated to Gain or Offset array, which can be used freely by application for instance to keep a trace of Gain or Offset table calibration conditions, either sensitivity, either focal plane array temperature.

8.5.2.8 eDALProxy640USBErr Proxy640USB_SaveBadPixels (HANDLE *paHandle,* const unsigned short ∗ *paTableX,* const unsigned short ∗ *paTableY,* unsigned short *paCount*)

Save bad pixel position into SmartIR640 slot data.

Parameters

in	*paHandle*	SmartIR640 handle.
in	*paTableX,pa-TableY*	Bad pixels position in image.
in	*paCount*	bad pixels count.

8.5.2.9 eDALProxy640USBErr Proxy640USB_LoadCurrentTableGain (HANDLE *paHandle,* unsigned char *paiIndex*)

Use SmartIR640 slot data as Gain values for NUC processing, i.e. retrieve it from SmartIR640, and set it to NUC processing.

Parameters

in	*paHandle*	SmartIR640 handle.
in	*paiIndex*	Slot index as data.

8.5.2.10 eDALProxy640USBErr Proxy640USB_LoadCurrentTableOffset (HANDLE *paHandle,* unsigned char *paiIndex*)

Use SmartIR640 slot data as Offset values for NUC processing, i.e. retrieve it from SmartIR640, and set it to NUC processing.

Parameters

in	*paHandle*	SmartIR640 handle.
in	*paiIndex*	Slot index as data.

8.5.2.11 eDALProxy640USBErr Proxy640USB_LoadCurrentBadPixels (HANDLE *paHandle*)

Use SmartIR640 stored bad pixel for bad pixel correction.

Parameters

in	*paHandle*	SmartIR640 handle.

8.5.2.12 eDALProxy640USBErr Proxy640USB_SaveCurrentTableGain (HANDLE *paHandle,* unsigned char *paiIndex,* const void ∗ *paData*)

Save current Gain values into SmartIR640 slot data.

Parameters

`in`	*paHandle*	SmartIR640 handle.
`in`	*paiIndex*	Slot index.
`in`	*paData*	Table associate data. NULL, or 60 bytes placeholder. paData is additional data associated to Gain or Offset array, which can be used freely by application for instance to keep a trace of Gain or Offset table calibration conditions, either sensitivity, either focal plane array temperature.

8.5.2.13 eDALProxy640USBErr Proxy640USB_SaveCurrentTableOffset (HANDLE *paHandle*, unsigned char *paiIndex*, const void $*$ *paData*)

Save Offset values into SmartIR640 slot data.

Parameters

`in`	*paHandle*	SmartIR640 handle.
`in`	*paiIndex*	Slot index.
`in`	*paData*	Table associate data. NULL, or 60 bytes placeholder. paData is additional data associated to Gain or Offset array, which can be used freely by application for instance to keep a trace of Gain or Offset table calibration conditions, either sensitivity, either focal plane array temperature.

8.5.2.14 eDALProxy640USBErr Proxy640USB_SaveCurrentBadPixels (HANDLE *paHandle*)

Save bad pixel position into SmartIR640 slot data.

Parameters

`in`	*paHandle*	SmartIR640 handle.

8.5.2.15 eDALProxy640USBErr Proxy640USB_SaveCurrentShutterlessTables (HANDLE *paHandle*)

Save Shutterless Tables into SmartIR640.

Parameters

`in`	*paHandle*	SmartIR640 handle.

Returns

eProxy640USBFeatureNotAvailable error if Shutterless is unavailable on the module.

8.5.2.16 eDALProxy640USBErr Proxy640USB_LoadCurrentShutterlessTables (HANDLE *paHandle*)

Load Shutterless Tables from SmartIR640 in order to use it with shutterless processing

Parameters

`in`	*paHandle*	SmartIR640 handle.

Returns

eProxy640USBFeatureNotAvailable error if Shutterless is unavailable on the module.

8.6 SmartIR640 Calibration

SmartIR640 NUC, bad pixel and Shutterless correction calibration. Refer to calibration example provided with the SDK for detailed usage of the following function.

Functions

- eDALProxy640USBErr Proxy640USB_AbortCalibration (HANDLE paHandle)
- eDALProxy640USBErr Proxy640USB_InitShutter2PtsCalibration (HANDLE paHandle, unsigned int iStage)
- eDALProxy640USBErr Proxy640USB_StepShutter2PtsCalibration (HANDLE paHandle, unsigned int iStage)
- eDALProxy640USBErr Proxy640USB_FinishShutter2PtsCalibration (HANDLE paHandle, unsigned int i-Stage)
- eDALProxy640USBErr Proxy640USB_InitShutterCalibration (HANDLE paHandle)
- eDALProxy640USBErr Proxy640USB_StepShutterCalibration (HANDLE paHandle)
- eDALProxy640USBErr Proxy640USB_FinishShutterCalibration (HANDLE paHandle)
- eDALProxy640USBErr Proxy640USB_InitSLCalibrationT0 (HANDLE paHandle, unsigned int iStage)
- eDALProxy640USBErr Proxy640USB_StepSLCalibrationT0 (HANDLE paHandle, unsigned int iStage)
- eDALProxy640USBErr Proxy640USB_FinishSLCalibrationT0 (HANDLE paHandle, unsigned int iStage)
- eDALProxy640USBErr Proxy640USB_InitSLCalibrationT1 (HANDLE paHandle)
- eDALProxy640USBErr Proxy640USB_StepSLCalibrationT1 (HANDLE paHandle)
- eDALProxy640USBErr Proxy640USB_FinishSLCalibrationT1 (HANDLE paHandle)

8.6.1 Detailed Description

SmartIR640 NUC, bad pixel and Shutterless correction calibration. Refer to calibration example provided with the SDK for detailed usage of the following function. NUC can be a two points calibration, or a one point calibration. Shutterless can be a T0 calibration only or T0 and T1 calibration.

This set of function provide 2 kind of NUC calibrations :

- Full Calibration.
- Fast Calibration.

And Shutterless Calibration compose of :

- T0 Calibration.
- T1 Calibration.

See Also

User's Guide or Calibration Process for details.

8.6.2 Function Documentation

8.6.2.1 eDALProxy640USBErr Proxy640USB_AbortCalibration (HANDLE *paHandle*)

Abort a Calibration process and reset the sequencing. None of the corrections table will be change by the abort calibration.

Parameters

in	*paHandle*	SmartIR640 Handle.

8.6.2.2 eDALProxy640USBErr Proxy640USB_InitShutter2PtsCalibration (HANDLE *paHandle*, unsigned int *iStage*)

Prepare NUC Calibration engine.

Parameters

in	*paHandle*	SmartIR640 Handle.
in	*iStage*	Stage of the calibration (1 or 2).

Returns

eProxy640USBSequencingError see Calibration Process for details.

8.6.2.3 eDALProxy640USBErr Proxy640USB_StepShutter2PtsCalibration (HANDLE *paHandle*, unsigned int *iStage*)

Add image for Shutter 2pts calibration.

Low temperature image for iStage = 1, High temperature image for iStage = 2.

Parameters

in	*paHandle*	SmartIR640 Handle.
in	*iStage*	Stage of calibration (1 (low) or 2 (high)).

Returns

eProxy640USBSequencingError see Calibration Process for details.

8.6.2.4 eDALProxy640USBErr Proxy640USB_FinishShutter2PtsCalibration (HANDLE *paHandle*, unsigned int *iStage*)

Perform two points calibration using low and high temperature images.

Once calibration is done, new Gain, Offset and bad pixel are set to current NUC and BPC processing.

Parameters

in	*paHandle*	SmartIR640 handle.
in	*iStage*	Stage of the calibration. If stage = 2, perform the final step of calibration. Once is done, new Gain, Offset and bad pixel are set to current NUC and BPC processing.

Returns

eProxy640USBSequencingError see Calibration Process for details.

8.6.2.5 eDALProxy640USBErr Proxy640USB_InitShutterCalibration (HANDLE *paHandle*)

Prepare Shutter Calibration engine, also called one point calibration.

This calibration will only produce new Offset values.

Parameters

in	*paHandle*	SmartIR640 handle.

Returns

eProxy640USBSequencingError see Calibration Process for details.

8.6.2.6 eDALProxy640USBErr Proxy640USB_StepShutterCalibration (HANDLE *paHandle*)

Add image to prepare Shutter Calibration

Parameters

in	*paHandle*	SmartIR640 handle.

Returns

eProxy640USBSequencingError see Calibration Process for details.

8.6.2.7 eDALProxy640USBErr Proxy640USB_FinishShutterCalibration (HANDLE *paHandle*)

Perform Shutter calibration.

Once calibration is done, Offset values are set to current NUC processing.

Parameters

in	*paHandle*	SmartIR640 handle.

Returns

eProxy640USBSequencingError see Calibration Process for details.

8.6.2.8 eDALProxy640USBErr Proxy640USB_InitSLCalibrationT0 (HANDLE *paHandle,* unsigned int *iStage*)

Initialise Stage for Shutterless Calibration T0

Must be use on correct sequencing with SmartIR640 shutterless module.

Parameters

in	*paHandle*	SmartIR640 handle.
in	*iStage*	Stage number of calibration.

Returns

eProxy640USBSequencingError see Calibration Process for details.

8.6.2.9 eDALProxy640USBErr Proxy640USB_StepSLCalibrationT0 (HANDLE *paHandle,* unsigned int *iStage*)

Add image for Shutterless Calibration T0

Must be used on correct sequencing with SmartIR640 shutterless module.

Parameters

`in`	*paHandle*	SmartIR640 handle.
`in`	*iStage*	Stage number of calibration (1 = low temperature, 2 = high temperature).

Returns

ePProxy640USBSequencingError see Calibration Process for details.

8.6.2.10 eDALProxy640USBErr Proxy640USB_FinishSLCalibrationT0 (HANDLE *paHandle,* unsigned int *iStage*)

Perform Shutterless T0 calibration

Must be used on correct sequencing with SmartIR640 shutterless module. Once calibration is done, new Shutterless tables are set to current Shutterless processing.

Parameters

`in`	*paHandle*	SmartIR640 handle.
`in`	*iStage*	Stage number of calibration (1 = low temperature, 2 = high temperature).

Returns

eProxy640USBSequencingError see Calibration Process for details.

8.6.2.11 eDALProxy640USBErr Proxy640USB_InitSLCalibrationT1 (HANDLE *paHandle*)

Initialise Shutterless Calibration T1

Must be use on correct sequencing with SmartIR640 shutterless module.

Parameters

`in`	*paHandle*	SmartIR640 handle.

Returns

eProxy640USBSequencingError see Calibration Process for details.

8.6.2.12 eDALProxy640USBErr Proxy640USB_StepSLCalibrationT1 (HANDLE *paHandle*)

Add image for Shutterless Calibration T1

Must be use on correct sequencing with SmartIR640 shutterless module.

Parameters

`in`	*paHandle*	SmartIR640 handle.

Returns

eProxy640USBSequencingError see Calibration Process for details.

8.6.2.13 eDALProxy640USBErr Proxy640USB_FinishSLCalibrationT1 (HANDLE *paHandle*)

Perform Shutterless Calibration T1

Must be use on correct sequencing with SmartIR640 shutterless module. Once Calibration is done, new Shutterless T1 tables are set for current Shutterless processing.

Parameters

`in`	*paHandle*	SmartIR640 handle.

Returns

eProxy640USBSequencingError see Calibration Process for details.

8.7 Function return code

Function execution returned code.

Enumerations

- enum eDALProxy640USBErr {
  eProxy640USBSuccess =0, eProxy640USBParameterError, eProxy640USBHandleError, eProxy640USBInitFailed,
  eProxy640USBOpenFailed, eProxy640USBCommFailed, eProxy640USBTimeout, eProxy640USBSyncBroken,
  eProxy640USBSequencingError, eProxy640USBFeatureNotAvailable, eProxy640USBBistInitFailure, eProxy640USBBistFailure,
  eProxy640USBFormatFailed, **eProxy640USBErrTotal** }

Functions

- const char ∗ Proxy640USB_GetErrorString (eDALProxy640USBErr paeError)

8.7.1 Detailed Description

Function execution returned code. eDALProxy640USBErr is return by most functions as a result of execution.

See Also

eDALProxy640USBErr() to convert code to user friendly string.

8.7.2 Enumeration Type Documentation

8.7.2.1 enum eDALProxy640USBErr

Code return by most functions about execution.

Enumerator

eProxy640USBSuccess Function call success.

eProxy640USBParameterError Function call with wrong parameter.

eProxy640USBHandleError Function call with wrong or invalid SmartIR640 handle.

eProxy640USBInitFailed Internal error occur.

eProxy640USBOpenFailed Open connection to SmartIR640 failed. Maybe already connected

eProxy640USBCommFailed Exchange with SmartIR640 failed.

eProxy640USBTimeout Operation on SmartIR640 timeout before completed.

eProxy640USBSyncBroken GetImage(), Sync with SmartIR640 broken.

eProxy640USBSequencingError Function call outside correct sequencing

eProxy640USBFeatureNotAvailable Feature not available on this module or can't be use due to present configuration.

eProxy640USBBistInitFailure Built-In Self Test initialisation failed.

eProxy640USBBistFailure SmartIR640 reported a Built-In Self Test error.

eProxy640USBFormatFailed Incompatible file format for SmartIR640.

8.7.3 Function Documentation

8.7.3.1 const char∗ Proxy640USB_GetErrorString (eDALProxy640USBErr *paeError*)

Convert eDALProxy640USBErr to user message.

Parameters

in	*paeError*	Function returns error code.

Returns

User error message from eDALProxy640USBErr.

Note

String is C-Style, i.e. Ascii with null terminate byte.

Index