



TouchPrint Live Scan Multi-Use Software Development Kit

Programmer's Manual

Version 7.11 – October, 2011

L-1 Identity Solutions – Biometrics Division**5705 W Old Shakopee Rd****Suite 100****Bloomington, MN 55437-3107**

Phone: (952) 932-0888

Technical Support**US Customers:** 1-888-HELP IDX (1-888-435-7439) *or*
1-925-945-5512**International Customers:** Contact your L-1 Identity Solutions Sales Representative for the
Service Number for your country.**E-Mail:** L1BDSupport@L1ID.com**Web Site:** www.l1id.com

Copyright© 2011 by L-1 Identity Solutions Incorporated

All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without prior written permission from L-1 Identity Solutions, Incorporated. Contact L-1 Identity Solutions for information on foreign rights.

Notice

L-1 Identity Solutions has obtained information contained in this document from sources believed to be reliable. L-1 Identity Solutions makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. L-1 Identity Solutions shall not be liable for errors contained herein or for incidental consequential damages in connection with the furnishing, performance, or use of this material.

All products, names and services are trademarks or registered trademarks of their respective companies.

The information contained in this document is subject to change without notice.

IMPORTANT NOTE: This document contains confidential information that belongs solely to L-1 Identity Solutions Incorporated. This document shall not be copied, delivered, transferred, made available, or otherwise disclosed by you to any third party without the advance written consent of L-1 Identity Solutions Incorporated.

Table of Contents

1.	Minimum System Requirements.....	1
1.1.	Hardware	1
1.2.	Software	1
2.	Introduction - Overview.....	2
2.1.	Connections.....	2
2.2.	Images	2
2.3.	Architecture	4
3.	Application Program Flow.....	6
3.1.	Device Selection	6
3.2.	Manual Fingerprint Capture	6
3.3.	Automatic Fingerprint Capture	6
4.	TP-LSMULTI Interface Library	8
4.1.	Application Build Instructions.....	8
4.1.1.	C++	8
4.1.2.	Microsoft .NET	8
4.2.	Interface Functions / Methods	8
4.2.1.	InitializeAPI	8
4.2.2.	TerminateAPI	8
4.2.3.	Get Device List	8
4.3.	Device Functions / Methods.....	9
4.3.1.	Open / Close Device	9
4.3.2.	Is Capture Type Supported.....	10
4.3.3.	Is Auto Capture Supported.....	10
4.3.4.	Is Device UI Supported.....	11
4.3.5.	Get Max Video Size	12
4.3.6.	Get Finished Image Size	13
4.3.7.	Get / Set Device UI	13
4.3.8.	Start Preview	14
4.3.9.	Start Preview DFC	17
4.3.10.	Capture Control.....	18
4.3.11.	Get Image Quality Data	19
4.3.12.	Query Switches	21
4.3.13.	Device Connected.....	22
4.3.14.	Get Device Info	22
4.3.15.	Reset Device	23
4.3.16.	Is Calibration Supported	23
4.3.17.	Is Calibrated	24
4.3.18.	Calibrate Scanner	24
4.3.19.	Get / Set Resolution.....	25
4.3.20.	Get / Set Max Hand Size.....	26
4.3.21.	Get / Set Slap Finger Thresholds.....	27
4.3.22.	Get / Set Slap Auto-Capture Timeouts.....	27
4.3.23.	Get / Set Slap Auto-Capture Primary Fingers.....	28
4.3.24.	Send Bulk Data.....	29
5.	TP-LSMULTI Supported Devices	30

6.	SDK File List	31
7.	Driver Installation	33
7.1.	USB 2.0 or IEEE 1394 OHCI Driver Installation	33
8.	Device Firmware Installation	34
8.1.	Firmware Update Utility	34
8.2.	Application Firmware Verification & Update.....	34
9.	Equalization Reference Images	35



Note:

The TP-LSMULTI-SDK libraries and associated device firmware are released as a package. In the release notes, the TP-LSMULTI-SDK version number and the firmware numbers are noted. It is strongly recommended that you maintain this release version number relationship to prevent compatibility issues.

1. Minimum System Requirements

1.1. Hardware

- 1.7 GHz Intel Pentium IV or other IBM-compatible processor
 - 256 MB RAM, 1024 x 768 resolution, 16-bit color
 - TP-LSMULTI Scanner
 - Universal Serial Bus (USB) - USB 2.0 High Speed Host Peripheral Controller
 - IEEE 1394 (FireWire™) – 400Mbps OHCI-compliant IEEE 1394 PC adapter
- Note:** TP-4100A-ED scanners require a 1394 adapter or hub that can provide 12VDC +/- 0.5 and a minimum of 1.5 Amps.

1.2. Software

- Microsoft Windows XP SP3 / Windows Vista / Windows 7
- Microsoft Visual Studio 2010
- [.NET] Microsoft .NET Framework 2.0

2. Introduction - Overview

This document describes how the TP-LSMULTI SDK can be used to integrate up to one USB and one IEEE1394, or two USB-attached TP-LSMULTI fingerprint scanning devices into a custom software application. See *TP-LSMULTI Supported Devices* for a list of supported devices. Hand scanner and handprint references apply to 'HA' suffixed devices only, palm references apply to 'PA' devices only.

2.1. Connections

TP-LSMULTI scanning devices may be connected to the host PC via the host system's USB 2.0 port, or an installed OHCI-compliant IEEE 1394 PC adapter, depending on its supported communications link.

The scanner and host PC operate as nodes on the communications bus. The bus connection conveys command and status messages between the host application and the scanner. It also transports other data, such as video packets that allow display of real-time images on a host system monitor. IEEE 1394-attached scanners require that the host system and the scanner be the only devices on the bus.

2.2. Images

A TP-LSMULTI scanner works in conjunction with a host application to produce images of various forms. These include fingerprints, four-finger slap prints, palm prints, and handprints. Images are two-dimensional arrays of bytes with a known width and height. Each byte represents a single pixel. In most images, the pixel values represent shades of gray. Lesser values represent darker shades and greater values represent lighter shades. In decorated images, which are the result of real-time quality analysis, some pixel values represent shades of red.

Image Terminology

An **equalization reference image** is an image modified for use with a scanner's equalization function. Equalization is the process that compensates for the illumination characteristic of a scanning unit. It maps the range of pixel values coming from a camera to the desired output range. For a hand-scanning unit, this image has only a single line. See *Equalization Reference Images* for information on how to install these images.

An **equalized image** is an image captured and processed using the equalization reference image.

A **decorated image** is an equalized image modified via real-time quality analysis to distinguish good and poor areas of a fingerprint. Pixels with even values (i.e. pixels with the least significant bit cleared) are good, and those with odd values are poor. The display of poor pixels should be in shades of red. Lesser values should be bright red and greater values should be pale pink or white. This can be accomplished by setting the red component to 255 while keeping the green and blue values equal in all of the odd entries of the decorated image RGB display palette. e.g. Grayscale color (101, 101, 101) should become (255, 101, 101).

A **finished image** is an image returned from scanning unit upon completion of all image processing.

Captured Images

Three kinds of images may be returned to an application during the capture process:

1. **Video image** – In Live Preview mode, a continuous stream of video images are returned for display to allow the operator to

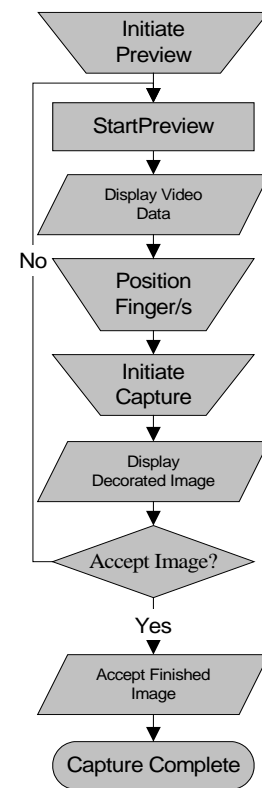


Figure 1: Image Capture

adjust the position of the fingers in the image and to watch a roll as the image is acquired.

2. **Decorated Image** – If the CMODE_RTQA capture mode is enabled, a single decorated image will be returned for operator review.
3. **Finished Image** – A final high quality image is returned as the final step of the capture process for additional processing, display, printing and/or saving.

There are five basic types of image captures:

1. Slap Capture – Single finger static placement
2. Four Finger Slap – Four finger static placement
3. Roll Capture – Single finger user-controlled left-to-right or right-to-left roll
4. Palm Scanner – Palm static placement
5. Hand Scanner – User-controlled drum hand scanner

Finished images can be captured at 500 dpi (dots-per-inch) or 1000 dpi, depending on the capabilities of the attached scanner, and have the following dimensions:

<u>Image Type</u>	<u>Resolution</u>	<u>Image Size</u>
Single Finger Roll/Slap	500 dpi	800 x 750
	1000 dpi	1600 x 1500
Four Finger/Two Thumb Slap	500 dpi	1600 x 1000 (<i>TP-3000A-ED</i>)
		1600 x 1200 (<i>TP-3800A-SD</i>)
		1600 x 1500 (<i>TP-4xxxA/UA</i>)
	1000 dpi	3200 x 2000 (<i>TP-3000A-ED</i>)
Palm Scanner	500 dpi	3200 x 1944 (<i>TP-5000A-HD</i>)
	1000 dpi	2500 x 2550
Hand Scanner	500 dpi	5000 x 5100 (<i>TP-5300A</i>)
		2576 x 3998 (max 8")
	1000 dpi	2576 x 4998 (max 10")
		5152 x 7996 (max 8")
		5152 x 9996 (max 10")

2.3. Architecture

The TP-LSMULTI SDK is designed to support either two USB 2.0, or one USB 2.0 and one IEEE 1394-attached TP-LSMULTI devices. IEEE 1394 devices may be connected via any OHCI-compliant IEEE 1394 FireWire™ PC adapter. There are multiple software component layers that run on the Microsoft Windows 2000/XP operating systems to support each pathway. The layers and their functions are described below.

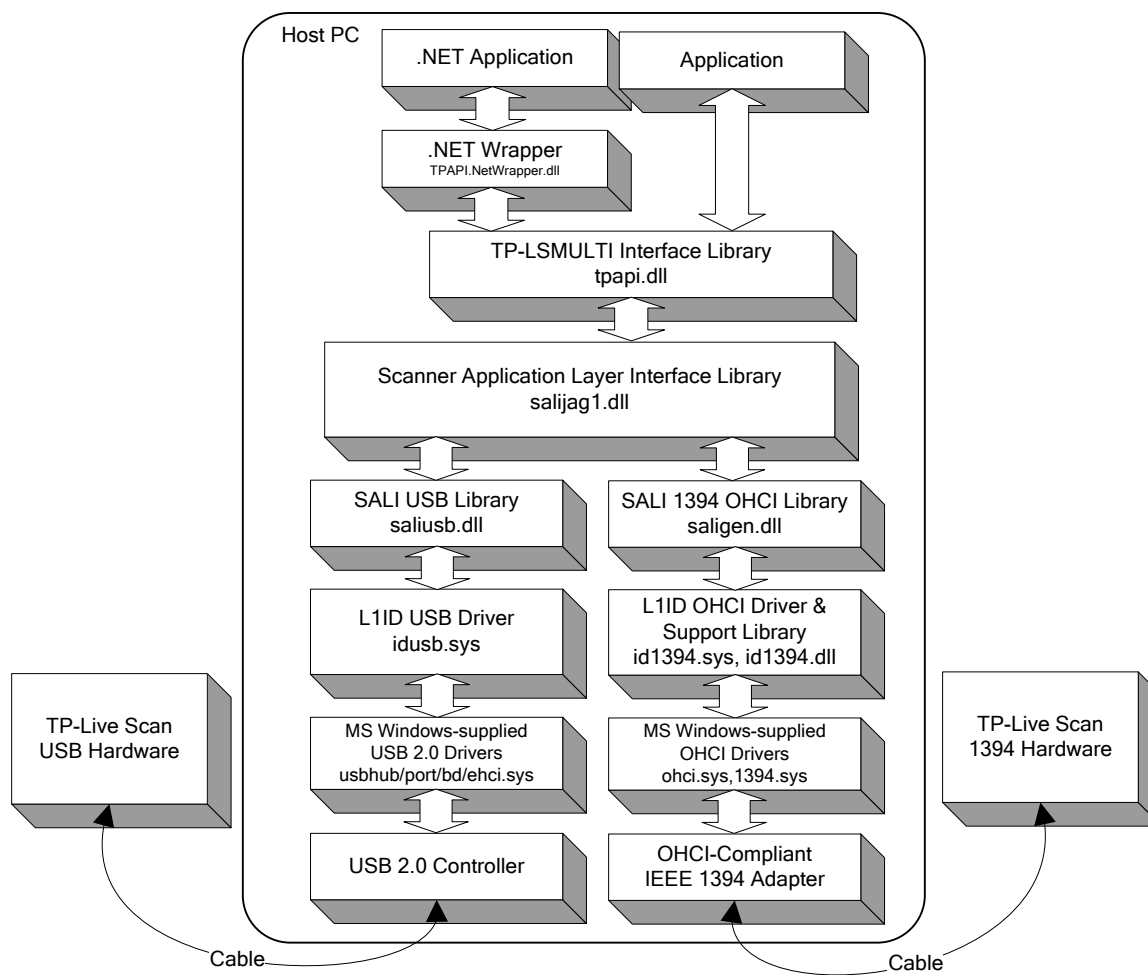


Figure 2: Block diagram of system architecture.

Application Layer

At the top is the Application Layer. Unmanaged applications interface directly with the TP-LSMULTI Interface Library. Microsoft .NET applications interface with the TP-LSMULTI Interface Library through the .NET Wrapper. Only one application may be active and connected to the device for it to function correctly. MFC, Visual Basic 6.0, and C# sample applications are provided to aid in understanding and programming to this interface.

.NET Wrapper

A .NET library wrapping the TP-LSMULTI Interface Library is provided for Microsoft .NET applications. It is the .NET software developer's interface to the TP-Live Scan device. The library consists of two classes; an interface class, and a device class. This document describes the properties and methods available through these classes which allow a .NET developer to create a custom application using TP-LSMULTI devices.

TP-LSMULTI Interface Library

The TP-LSMULTI Interface Library is the TP-LSMULTI Application Programming Interface (TPAPI) library. It is the software developer's interface to the TP-Live Scan device. This document describes the functions that it provides to the developer for creating a custom application to use with TP-LSMULTI devices.

Scanner Application Layer Interface Libraries

The Scanner Application Layer Interface (SALI) is comprised of several libraries providing low-level API scanner communication functions. These libraries provide the command and status message functions used by the TP-LSMULTI Interface Library to communicate with a scanner.

The role of salijag1.dll is to identify and direct flow to and from the active link-layer. The salijag1.dll library will scan for a device on all supported links until it identifies a TP-LSMULTI SDK-supported device and direct all traffic through the link as long as it remains active.

Saliusb.dll provides the top-level interface to the USB 2.0 bus and saligen.dll to a generic OHCI-compliant IEEE 1394 FireWire™ adapter.

Device Driver Layer

The Device Driver layer is the Windows kernel-mode layer and its associated user-mode libraries.

The TP-LSMULTI-SDK USB 2.0 driver works in conjunction with Microsoft-supplied bus and port drivers to negotiate and control communication between the host PC and the scanner over the Universal Serial Bus (USB). Image data transfer bandwidth requirements necessitate that the host USB controller support USB 2.0 High-Speed mode. Attempts to communicate with an attached scanner through a USB 1.1 are unreliable.

The TP-LSMULTI-SDK IEEE 1394 Open Host Controller Interface (OHCI) driver and its support library also work in conjunction with Microsoft-supplied IEEE 1394 bus and port drivers to provide a communications channel between the host PC and the scanner over the FireWire™ interface. An OHCI-compliant IEEE 1394 adapter will need to be installed in the host system.

When an active scanner is connected to one of the supported communications links, Windows will notify the user that new hardware has been found, and the appropriate device driver will then be loaded. This driver will handle the specific details of the interface and support calls made by the SALI layer.

3. Application Program Flow

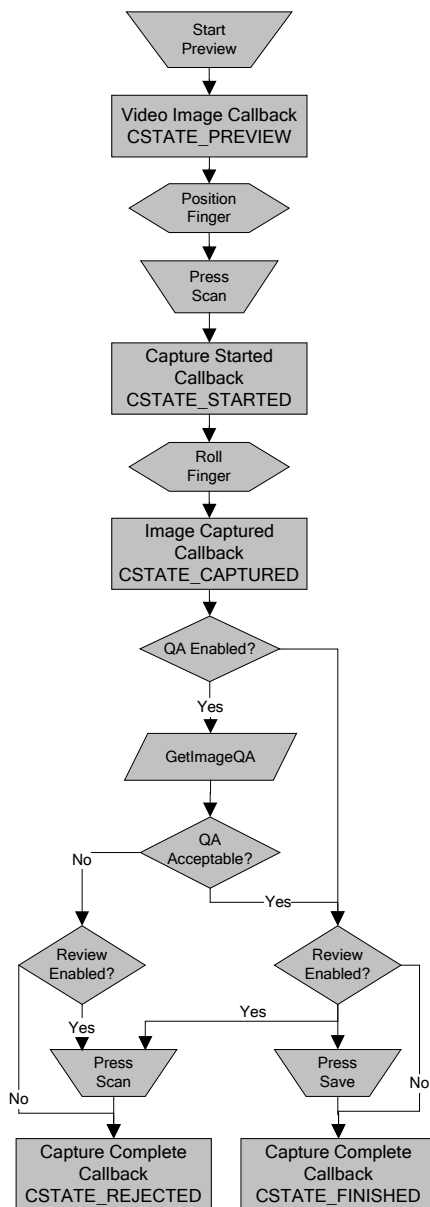


Figure 3: Manual Fingerprint Capture

3.1. Device Selection

Prior to initiating any device-specific commands, the application must select and open an attached device. The device handle returned in the *Open Device* call should then be used in all subsequent device communication.

3.2. Manual Fingerprint Capture

Manual fingerprint captures are controlled by an operator either through the application's user interface (see *Capture Control*) or buttons on the Live Scan device.

The following illustrates a typical manual fingerprint capture:

1. Start a preview of the desired type by calling the *Start Preview* method. This method will have the type and mode of capture passed in as a parameter as well as an image callback function to transfer the image data (live video, decorated images, and finished images) to the application for display, printing, and saving.
2. On Live Scan devices with Finger Contact LEDs, the LEDs corresponding to the selected capture type will go red, indicating to the operator which finger or fingers to place on the platen. As the scanner detects finger contact the LED color will change to yellow. When an acceptable threshold of contact is detected (or 2 seconds have elapsed) the LED will turn green indicating that the capture may commence.
3. The fingerprint image capture may then be triggered by the operator pressing the device's SCAN button (*if available*) or an application-provided GUI SCAN button. The application callback function will be signaled (CSTATE_STARTED) allowing it to give the operator feedback on the capture. The device will then scan the fingerprint image.
4. Upon completion of the capture, information regarding the quality of the capture, as well as a decorated image may then be available to the application for evaluation.

5. Following the application's evaluation of the image quality information (if enabled), the operator may visually review the decorated image, and either re-Scan the image, or if it meets the application-defined quality criteria, Save it (see *Get Image Quality Data*).
6. Saving the image will initiate transfer to the application of the final finished image via the image callback function designated in the *Start Preview* call.

3.3. Automatic Fingerprint Capture

TP-LSMULTI Live Scan devices are also able to trigger image captures automatically on some or all fingerprint capture types (see *Is Auto Capture Supported*). Two styles of auto-capture are available:

- Ink-like Auto-capture

- Auto-capture with Preview (*rolls only*)

Ink-like Auto-capture

This capture mode is designed to simulate a slap or rolled finger print capture using ink.

The operator action and Finger Contact LED color sequence is the same as a manual capture up to the point where the finger LEDs turn green. At this point, the scanner will emit a single beep, and automatically begin capturing a final image (CSTATE_STARTED). The operator should then roll the finger across the platen (rolls) and/or lift the finger from the platen at the next beep. Upon completion of the capture (CSTATE_CAPTURED) the finger LEDs will turn off.

Auto-capture with Preview

This auto-capture mode is designed to enable an operator to find and center the subject's fingerprint core on the platen, while still allowing a button-free capture of a rolled fingerprint.

The operator action and Finger Contact LED color sequence is the same as a manual capture up to the point where the finger LED turns yellow. At this point, the operator should center the fingerprint core on the platen, rock the finger to either side, and hold there until the finger LED goes green (indicating that acceptable contact is detected, or 2 seconds have elapsed). When the scanner emits a single beep indicating that the final capture has begun (CSTATE_STARTED), the operator should roll the finger across the platen. Upon completion of the capture (CSTATE_CAPTURED) the finger LEDs will turn off.

4. TP-LSMULTI Interface Library

Commands will return TPAPI_NOERROR if successful.

4.1. Application Build Instructions

4.1.1. C++

C++ applications should be built using the SDK-supplied *tpapi.h* and *tpapi.lib* files.

4.1.2. Microsoft .NET

Microsoft .NET applications should add a reference to the SDK-supplied *TPAPI.NetWrapper* assembly. This assembly has been built with Strong-Name signing, possibly requiring an application rebuild should the version change.

4.2. Interface Functions / Methods

These functions and methods operate on the TouchPrint Interface and are not device-specific. Microsoft .NET methods operate on a *TpApi* interface object.

4.2.1. InitializeAPI

[C++] long STDCALL TP_InitializeAPI(void)

[.NET] TpApiError TpApi_InitializeAPI()

Initializes the TPAPI library. This will not reset, or otherwise alter the state of the scanner. It must be called prior to any other API functions.

Error Codes:

TPAPI_INVALIDSTATE

Initialize has already been performed

4.2.2. TerminateAPI

[C++] long STDCALL TP_TerminateAPI(void)

[.NET] TpApiError TpApi_TerminateAPI()

Close all open devices, terminate the TPAPI library, and return to the uninitialized state.

Error Codes:

TPAPI_INVALIDSTATE

API has not been initialized

4.2.3. Get Device List

[C++] long STDCALL TP_GetDeviceList(U32 *numDev, struct s_tpDevList *tpApiDevice)

[.NET] TpApiError TpApi_GetDeviceList (ref U32 numDev, ref TpApiDevice[] tpApiDevice)

Parameters:

numDev

[IN] Maximum number of devices to search for

[OUT] Number of devices found

tpApiDevice

struct s_tpDevList

```

{
    TCHAR devName [255];           // OS-registered device name
    char assy_model [16];          // Assembly Model (e.g. "TP4100")
    char assy_sn [16];             // Assembly Serial Number (e.g. "123456")
    U32 libType;                   // Communications link library type
}

```

[IN] Buffer in which to return an array of attached devices. This buffer must be large enough to hold the requested number of devices to search for (*numDev*).

[OUT] Array of information on available devices

This function will search for attached and available TP-LSMULTI SDK-supported devices, and return information required to select and open (see *Open Device*) them for communication.

Error Codes:

TPAPI_UNINITIALIZED

Library has not been successfully initialized

TPAPI_PARAMETEROUTOFRANGE

numDev or *tpApiDevice* is a NULL pointer

TPAPI_OUTOFRESOURCES

Memory or other OS resource allocation error

4.3. Device Functions / Methods

These functions and methods operate individual TouchPrint devices as specified by the device name, handle, or [.NET] *TpApiDevice* object.

4.3.1. Open / Close Device

[C++] long STDCALL TP_OpenDevice(TCHAR *devName, U32 libType, U32 *hDev)

[C++] long STDCALL TP_CloseDevice(U32 hDev)

[.NET] TpApiError OpenDevice()

[.NET] TpApiError CloseDevice()

Parameters:

devName

Operating system-registered device name

libType

Communications link library type

hDev

Device handle

Open or close device communications. The *Open Device* parameters may be obtained from *Get Device List*. The returned device handle should be used for all subsequent device API calls.

Error Codes:

TPAPI_UNINITIALIZED

Library has not been successfully initialized

TPAPI_PARAMETEROUTOFRANGE

devName or *hDev* is a NULL pointer, or *hDev* is an invalid handle

TPAPI_TOOMANYDEVICES

Communications link device limit already reached

TPAPI_OUTOFRESOURCES

Memory or other OS resource allocation error

4.3.2. Is Capture Type Supported

[C++] long STDCALL TP_IsCaptureTypeSupported(U32 hDev, enum eCaptureType capType, BOOL *result)

[.NET] TpApiError IsCaptureTypeSupported(eCaptureType capType, ref bool result)

Parameters:

hDev

Device handle

capType

enum eCaptureType:

- | | |
|----------------------|----------------------------|
| ▪ CTYPE_ROLL_LTHUMB | - Left Thumb roll |
| ▪ CTYPE_ROLL_LINDEX | - Left Index finger roll |
| ▪ CTYPE_ROLL_LMIDDLE | - Left Middle finger roll |
| ▪ CTYPE_ROLL_LRING | - Left Ring finger roll |
| ▪ CTYPE_ROLL_LLITTLE | - Left Little finger roll |
| ▪ CTYPE_ROLL_RTHUMB | - Right Thumb roll |
| ▪ CTYPE_ROLL_RINDEX | - Right Index finger roll |
| ▪ CTYPE_ROLL_RMIDDLE | - Right Middle finger roll |
| ▪ CTYPE_ROLL_RRING | - Right Ring finger roll |
| ▪ CTYPE_ROLL_RLITTLE | - Right Little finger roll |
| ▪ CTYPE_SLAP_LTHUMB | - Left Thumb slap (plain) |
| ▪ CTYPE_SLAP_RTHUMB | - Right Thumb slap (plain) |
| ▪ CTYPE_SLAP_FOUR | - Four-finger slap |
| ▪ CTYPE_SLAP_LFOUR | - Left four-finger slap |
| ▪ CTYPE_SLAP_RFOUR | - Right four-finger slap |
| ▪ CTYPE_SLAP_THUMBS | - Both thumbs slap |
| ▪ CTYPE_SLAP_LPALM | - Left palm-print |
| ▪ CTYPE_SLAP_RPALM | - Right palm-print |
| ▪ CTYPE_HAND | - Hand-print |

result

Boolean value indicating the capture type support

Returns an indication of support for the given capture type on the attached device.

Error Codes:

TPAPI_UNINITIALIZED

Library has not been successfully initialized

TPAPI_PARAMETEROUTOFRANGE

hDev is an invalid handle or *result* is NULL

TPAPI_DEVICEUNAVAILABLE

Device not found on the bus

TPAPI_BADRESPONSE

Unexpected or invalid response to device command

TPAPI_TIMEOUT

Timeout attempting to communicate with the device

4.3.3. Is Auto Capture Supported

[C++] long STDCALL TP_IsAutoCaptureSupported(U32 hDev, enum eCaptureType capType, BOOL *result)

[.NET] TpApiError IsAutoCaptureSupported(eCaptureType capType, ref bool result)

Parameters:

`hDev`

Device handle

`capType`

See *Is Capture Type Supported*

`result`

Boolean value indicating auto-capture support

Returns an indication of support for auto-capture of the given capture type on the device. If a device supports auto-capture for a given capture type, it will support all types of auto-capture for that type. See the *Start Preview capMode* parameter for a description of auto-capture.

Error Codes:

`TPAPI_UNINITIALIZED`

Library has not been successfully initialized

`TPAPI_PARAMETEROUTOFRANGE`

hDev is an invalid handle or *result* is NULL

`TPAPI_DEVICEUNAVAILABLE`

Device not found on the bus

`TPAPI_BADRESPONSE`

Unexpected or invalid response to device command

`TPAPI_TIMEOUT`

Timeout attempting to communicate with the device

4.3.4. Is Device UI Supported

[C++] `long STDCALL TP_IsDeviceUISupported(U32 hDev, enum eUIType UIType, BOOL *result)`

[.NET] `TpApiError IsDeviceUISupported(eUIType UIType, ref bool result)`

Parameters:

`hDev`

Device handle

`UIType`

- `UI_4_4_2_TRI_LED` - 4x4x2 tri-colored LED UI

`result`

Boolean value indicating user interface type support

Returns an indication of support for the given user interface on the device.

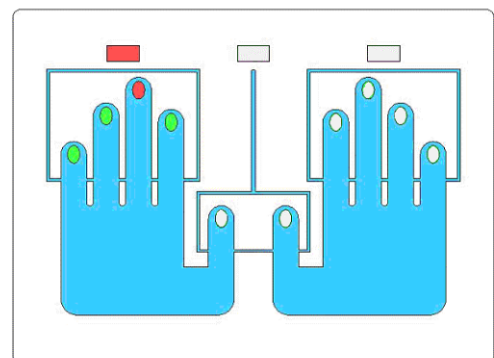
UI_4_4_2_TRI_LED User Interface

The 4x4x2 device user interface is supported on TP-4000 and TP-4100 Live Scan devices.

All LEDs are tri-color: red, yellow, and green.

Capture Status: The three LEDs across the top of the user interface are intended to indicate to the operator the type and status of the current capture. The application is responsible for setting the state of these LEDs (see *Set Device UI*).

As an example, an application might set the capture status LED to yellow initially, to indicate the group of



fingers that will be captured. Upon completion of the capture, it might set the LED to green to indicate success, or red for failure.

Finger Contact: The LEDs within each finger group are finger contact LEDs. They are grouped into 3 sections corresponding to the supported capture types: left 4-finger, right 4-finger and two thumbs. These LEDs indicate to the user the contact status currently detected by the device for that finger. Red will indicate poor (or no) contact, yellow partial contact, and green good contact. During the capture process, Finger Contact LEDs are under device control and should not be modified by the application.

Error Codes:

TPAPI_UNINITIALIZED

Library has not been successfully initialized

TPAPI_PARAMETEROUTOFRANGE

hDev is an invalid handle or *result* is NULL

TPAPI_DEVICEUNAVAILABLE

Device not found on the bus

TPAPI_BADRESPONSE

Unexpected or invalid response to device command

TPAPI_TIMEOUT

Timeout attempting to communicate with the device

4.3.5. Get Max Video Size

[C++] long STDCALL TP_GetMaxVideoSize(U32 hDev, enum eCaptureType capType, U16 *maxW, U16 *maxH)

[.NET] TpApiError GetMaxVideoSize(eCaptureType capType, ref Size maxVideo)

Parameters:

hDev

Device handle

capType

See *Is Capture Type Supported*

maxW, maxH, maxVideo

Maximum width and height of the image returned in the CSTATE_PREVIEW and CSTATE_CAPTURED callback functions for the attached device.

Returns the maximum size of the video image that the device is capable of returning in CSTATE_PREVIEW mode for the given capture type.

Error Codes:

TPAPI_UNINITIALIZED

Library has not been successfully initialized

TPAPI_PARAMETEROUTOFRANGE

hDev is an invalid handle, or *maxW*, *maxH*, or *maxVideo* is NULL

TPAPI_DEVICEUNAVAILABLE

Device not found on the bus

TPAPI_BADRESPONSE

Unexpected or invalid response to device command

TPAPI_TIMEOUT

Timeout attempting to communicate with the device

4.3.6. Get Finished Image Size

[C++] long STDCALL TP_GetFinishedImageSize(U32 hDev, enum eCaptureType capType, U32 dpi, struct s_imgSize *imgSize)

[.NET] TpApiError GetFinishedImageSize(eCaptureType capType, U32 dpi, ref Size imgSize)

Parameters:

hDev

Device handle

capType

See *Is Capture Type Supported*

dpi

Resolution, in dots-per-inch, of the finished image; Valid values = 500 or 1000

imgSize

Image width and height

Gets the size of the finished image that will be returned in the CSTATE_FINISHED callback function for the given capture type and resolution.

Error Codes:

TPAPI_UNINITIALIZED

Library has not been successfully initialized

TPAPI_PARAMETEROUTOFRANGE

hDev is an invalid handle, or *dpi* value invalid or not supported or *imgSize* is NULL

TPAPI_DEVICEUNAVAILABLE

Device not found on the bus

TPAPI_BADRESPONSE

Unexpected or invalid response to device command

TPAPI_TIMEOUT

Timeout attempting to communicate with the device

4.3.7. Get / Set Device UI

[C++] long STDCALL TP_GetDeviceUI(U32 hDev, enum eUIType UIType, void *UIData)

[C++] long STDCALL TP_SetDeviceUI(U32 hDev, enum eUIType UIType, void *UIData)

[.NET] TpApiError GetDeviceUI(eUIType UIType, s_442TriLED UIData)

[.NET] TpApiError SetDeviceUI(eUIType UIType, ref s_442TriLED UIData)

Parameters:

hDev

Device handle

UIType

- UI_4_4_2_TRI_LED - 4x4x2 tri-colored LED UI

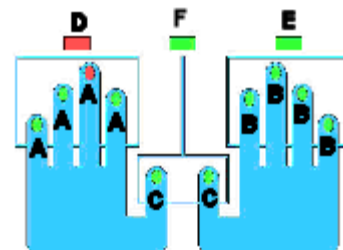
UIData

User interface structure buffer

UI 4 4 2 TRI LED User Interface

struct s_442TriLED

```
{
    enum eLEDState left_four_progress;    [D]
    enum eLEDState right_four_progress;   [E]
    enum eLEDState thumb_progress;        [F]
```



```

enum eLEDState left_index;      [A]
enum eLEDState left_middle;    [A]
enum eLEDState left_ring;      [A]
enum eLEDState left_little;    [A]
enum eLEDState right_index;    [B]
enum eLEDState right_middle;   [B]
enum eLEDState right_ring;     [B]
enum eLEDState right_little;   [B]
enum eLEDState left_thumb;     [C]
enum eLEDState right_thumb;    [C]
}
enum eLEDState
{
    • UI_LED_OFF                - Tri-colored LED off
    • UI_LED_GREEN              - Tri-colored LED green
    • UI_LED_RED                 - Tri-colored LED red
    • UI_LED_YELLOW              - Tri-colored LED yellow
}

```

Set and get the current state of the user interface. *Is Device UI Supported* can be used to determine what, if any, user interface is present in the device. Note that during the capture process, Finger Contact LEDs are under device control and should not be modified by the application.

Error Codes:

TPAPI_UNINITIALIZED

Library has not been successfully initialized

TPAPI_PARAMETEROUTOFRANGE

hDev is an invalid handle, or *UIData* is NULL

TPAPI_DEVICEUNAVAILABLE

Device not found on the bus

TPAPI_BADRESPONSE

Command or UIType not supported by device

TPAPI_TIMEOUT

Timeout attempting to communicate with the device

4.3.8. Start Preview

[C++] long STDCALL TP_StartPreview(U32 hDev, enum eCaptureType capType, U32 capMode, U16 videoW, U16 videoH, long (__stdcall *TP_ImgCallbk)(U32 hDev, S32, U8 *, U16, U16, void *pCtxt), void *pCtxt)

[.NET] TpApiError StartPreview(eCaptureType capType, eCaptureMode capMode, Size videoSz)

Parameter:

hDev

Device handle

capType

enum eCaptureType:

- CTYPE_ROLL_LTHUMB - Left Thumb roll
- CTYPE_ROLL_LINDEX - Left Index finger roll
- CTYPE_ROLL_LMIDDLE - Left Middle finger roll
- CTYPE_ROLL_LRING - Left Ring finger roll
- CTYPE_ROLL_LLITTLE - Left Little finger roll
- CTYPE_ROLL_RTHUMB - Right Thumb roll

- CTYPE_ROLL_RINDEX - Right Index finger roll
- CTYPE_ROLL_RMIDDLE - Right Middle finger roll
- CTYPE_ROLL_RRING - Right Ring finger roll
- CTYPE_ROLL_RLITTLE - Right Little finger roll
- CTYPE_SLAP_LTHUMB - Left Thumb slap (plain)
- CTYPE_SLAP_RTHUMB - Right Thumb slap (plain)
- CTYPE_SLAP_LFOUR - Left four-finger slap
- CTYPE_SLAP_RFOUR - Right four-finger slap
- CTYPE_SLAP_THUMBS - Both thumbs slap
- CTYPE_SLAP_LPALM - Left palm-print
- CTYPE_SLAP_RPALM - Right palm-print
- CTYPE_HAND - Hand-print

capMode

The capture mode bits both define how the capture process will proceed as well as whether real-time quality analysis (RTQA) will take place on the captured image. Below are the bit definitions for the capture mode.

- CMODE_RTQA** Enable RTQA and create a decorated image in the device. RTQA information will be available only during the CSTATE_CAPTURED via the *Get Image Quality Data* function. See *Images* for information on displaying decorated images.
- CMODE_REVIEW** Suspends the capture process following retrieval of the image quality information and decorated image (if enabled), allowing an operator to decide whether to accept or reject the captured image. The capture process will wait in this decision stage until an operator presses one of the scanner buttons. The SCAN button rejects the image and the SAVE button accepts the image. If the image has been rejected based on image quality criteria (see *Get Image Quality Data*), only the SCAN button will be enabled.
- CMODE_AUTO_CAP** Enable Ink-like Auto-capture mode. If enabled, the device is responsible for determining when to trigger a capture based on the degree of finger contact detected. See *Ink-like Auto-capture* for a description of a capture operation utilizing this mode of operation.
- CMODE_AUTO_CAP_WITH_PREVIEW** Enable Auto-capture with Preview mode. This mode is valid only on roll *capTypes* and cannot be set with CMODE_AUTO_CAP. If enabled, the device is responsible for determining when to trigger a capture. See *Auto-capture with Preview* for a description of a capture operation utilizing this mode of operation.
- CMODE_DISABLE_KEYS** Disable the device's SCAN/SAVE buttons (if available) during the capture process.
- CMODE_DISABLE_BEEP** Disable device-controlled beeps during a capture operation.

videoW, videoH, videoSz

Width and height of the image returned in the CSTATE_PREVIEW and CSTATE_CAPTURED callback functions. The width should be a multiple of 4. The maximum value can be obtained via *Get Max Video Size*.

[C++] long __stdcall *TP_ImgCallbk(U32 hDev, U32 capState, U8 *imgBuf, U16 imgWdth, U16 imgHght, void *pCtxt)

[.NET] delegate int TP_ImgCallbk(eCaptureState capState, byte[] imgBuf, Size imgSz)

Application-provided callback function that will be called when either a frame is received or the

capture state has changed. For .NET applications, this callback is a property which should be set prior to calling *Start Preview*.

hDev is the handle to the device initiating the callback. The *capState* will indicate the capture process state code, and possibly the type image in *imgBuf* (see *Images*). *pCtxt* is the application-defined value passed into *Start Preview*. Applications should avoid blocking in a callback function since it must complete before any subsequent callbacks can occur. Also, no TPAPI functions should be called from CSTATE_PREVIEW, CSTATE_STARTED, and non-RTQA CSTATE_CAPTURED callbacks.

Valid *capStates* are:

- CSTATE_PREVIEW - Preview image available
- CSTATE_STARTED - Operator has pressed SCAN button, image capture has begun
- CSTATE_CAPTURED - Capture complete, a decorated image may be available

Final capture process states:

- CSTATE_ABORTED - SAVE button pressed during Preview stage
- CSTATE_FINISHED - Finished image available, capture process complete
- CSTATE_REJECTED - SCAN button pressed during Review stage
- CSTATE_TOOFAST - Hand scanner drum rolled too fast or too slow, capture failed.
- CSTATE_CANCELED - Capture stopped by the host PC
- CSTATE_FAILED - Failed due to hardware or system error. The recommended recovery mechanism is to:
 1. Retry the capture.
 2. If the retry capture fails, then call *Reset Device*, and retry the capture once more.
 3. If the capture still fails, re-boot the entire system.

imgWidth, *imgHght*, and *imgSz* indicate the size of the image in *imgBuf* and will be valid only for the duration of the callback. See 2.2 for image sizes. A return code of zero will indicate acceptance of the image, -1 rejection, and is only valid on CSTATE_CAPTURED (decorated image) callbacks (see *Get Image Quality Data*).

pCtxt

Application-defined value passed back in *TP_ImgCallbk*.

Start Preview will initiate Preview Mode (Live Video) and indicate the type of image that will be captured when the operator presses the SCAN button on the device. Preview Mode is intended to provide visual feedback to the operator for both pre-capture finger positioning and rolling prints. Image capture buffers will be allocated and the Live Scan hardware will be instructed to start streaming live preview images (video) to the computer. Some live images will be down-sampled and contrast-adjusted at the scanner to conserve bandwidth while maintaining maximum image quality. The application-supplied *TP_ImgCallbk* function will be called with received preview images as well the decorated and finished images. See *Application Program Flow* for a description of the image capture process.

Hand Scanner-Only:

Hand-scanner preview images will become available only AFTER a capture has been initiated by the operator (i.e. SCAN button pressed) and will return a series of four-line partial frames to be aggregated and displayed by the application. The *imgHght* parameter will indicate the vertical position within the image of the first line in each returned *imgBuf*. A vertical position of zero would indicate the bottom-most line of the image. Successive lines should be displayed bottom-to-top. The aggregate size of the completed preview image will depend on the length of the hand rolled.

Hand-scanner preview images also contain embedded drum speed information in the form of overlays on the left and the right edges. Each overlay region contains values that should be set as colors in the display palette whose values roughly indicate the speed of drum rotation. There are four possible pixel values in the overlay regions:

- 0 - means that the drum is rotating too fast or too slow and that the scanner will refuse the image (CSTATE_TOOFAST). These pixels should normally be displayed as red.
- 1 - means that the drum is rotating faster than desired, but the scanner will still allow the image. These pixels should normally be displayed as yellow.
- 2 - means that the drum is rotating at an acceptable speed. These pixels should normally be displayed as green.
- 3 - means that the drum is rotating slower than desired, but the scanner will still allow the image. These pixels should normally be displayed as blue.

When a host application configures a palette to display these values as colors, it can assume that these pixel values will not be present in the non-overlaid regions of the display.

Error Codes:

TPAPI_UNINITIALIZED

Library has not been successfully initialized

TPAPI_PARAMETEROUTOFRANGE

hDev is an invalid handle, or invalid *capType*, *capMode*, *videoW*, or *videoH* specified

TPAPI_DEVICEUNAVAILABLE

Device not found on the bus

TPAPI_INVALIDSTATE

Command cannot be completed in the current capture state

TPAPI_TIMEOUT

Timeout attempting to communicate with the device

TPAPI_BADRESPONSE

Unexpected or invalid response to device command; a device reset may be required

TPAPI_BADFILE

Error reading equalization reference images

TPAPI_OUTOFRESOURCES

Memory or other OS resource allocation error

4.3.9. Start Preview DFC

[C++] long STDCALL TP_StartPreviewDFC(U32 hDev, enum eCaptureType capType, U32 capMode, U32 dfcMap, s_imgSize *videoSz, long (__stdcall *TP_ImgCallbk)(U32 hDev, S32, U8 *, U16, U16, void *pCtxt), void *pCtxt)

[.NET] TpApiError StartPreviewDFC(eCaptureType capType, eCaptureMode capMode, U32 dfcMap, Size videoSz)

Parameter:

capType

- CTYPE_SLAP_FOUR - Four-finger slap

dfcMap

A bitmapped array of fingers to be captured. The allowable fingers and the maximum number of fingers are determined by the *capType*.

Allowable fingers:

CTYPE_SLAP_FOUR - the maximum number of fingers that may be selected for capture is 4 and the thumb may not be selected if fingers from that same hand are selected.

- BIT_RTHUMB - Right Thumb
- BIT_RINDEX - Right Index
- BIT_RMIDDLE - Right Middle
- BIT_RRING - Right Ring
- BIT_RLITTLE - Right Little

- BIT_LTHUMB - Left Thumb
- BIT_LINDEX - Left Index
- BIT_LMIDDLE - Left Middle
- BIT_LRING - Left Ring
- BIT_LLITTLE - Left Little

`hDev`, `capMode`, `videoSz`, `TP_ImgCallbk`, `pCtxt`

See *Start Preview*.

The ***Start Preview DFC*** function is a Directed Finger Capture (DFC) version of *Start Preview*. DFC allows a caller to specify which individual fingers are to be captured. The device UI (if present) will prompt for the requested fingers and auto-capture triggering will be based on the number and the individual thresholds of those fingers (see *Get / Set Slap Finger Thresholds*). For a more complete description of the image capture process see *Start Preview*.

Error Codes:

TPAPI_UNINITIALIZED

Library has not been successfully initializedTPAPI_PARAMETEROUTOFRANGE

hDev is an invalid handle, or invalid *capType*, *capMode*, or *videoSz* specified

TPAPI_DFC_TOOMANYFINGERS

Too many fingers selected in *dfcMap*

TPAPI_DFC_NONPERMISSABLE_FINGER_COMBINATION

The thumb may not be selected in *dfcMap* if fingers from that same hand are selected

TPAPI_DEVICEUNAVAILABLE

Device not found on the bus

TPAPI_INVALIDSTATE

Command cannot be completed in the current capture state

TPAPI_TIMEOUT

Timeout attempting to communicate with the device

TPAPI_BADRESPONSE

Unexpected or invalid response to device command; a device reset may be required

TPAPI_BADFILE

Error reading equalization reference images

TPAPI_OUTOFRESOURCES

Memory or other OS resource allocation error

4.3.10. Capture Control

[C++] `long STDCALL TP_CaptureControl(U32 hDev, enum eCapControl capCtl)`

[.NET] `TpApiError CaptureControl(eCapControl capCtl)`

Parameters:

`hDev`

Device handle

`capCtl`

- CC_CANCEL_CAPTURE - Abort the capture process. This command will block until the CSTATE_CANCELED *TP_ImgCallbk* function signals that the device has acknowledged the command.
- CC_CAPTURE_IMAGE - Trigger an image capture (SCAN).
- CC_FINISH_IMAGE - Create a finished image (SAVE). To be used during the operator review stage.

This function provides the mechanism for an application to control the flow of a capture process.

Error Codes:

TPAPI_UNINITIALIZED

Library has not been successfully initialized

TPAPI_PARAMETEROUTOFRANGE

hDev is an invalid handle

TPAPI_DEVICEUNAVAILABLE

Device not found on the bus

TPAPI_BADRESPONSE

Unexpected or invalid response to device command

TPAPI_TIMEOUT

Timeout attempting to communicate with the device

4.3.11. Get Image Quality Data

[C++] long STDCALL TP_GetImageQA(U32 hDev, struct s_imgQA *sParams)

[.NET] TpApiError GetImageQA(ref s_imgQA sParams)

Parameter:

hDev

Device handle

sParams

struct s_imgQA

{

U16 cover_factor; // roll coverage factor (times 100)

U16 short_roll_time; // number of milliseconds to roll

U16 short_contact_area; // % area of contact

U16 good_area; // % area of good contrast

U16 dark_area; // % area too dark

U16 light_area; // % area too light

U16 smear_area; // % area too smeared

}

Buffer in which to return the image quality parameters

Roll-Only Parameters:

For finger rolls, the coverage factor is the ratio of the total width of a roll and the plain width of the finger, expressed as a percentage. The widest measured contact in any one video frame, which usually occurs near the middle of the roll, is the plain width. The extremes of contact measured in the first and last frames determine the total width of the roll. A host application can use the coverage factor to decide whether an operator rolled the finger enough. Its value cannot be less than 100. It is typically less than 250, although a very wide roll with a narrow finger may be larger. Suggested Default Minimum Finger Coverage Factor = 125.

For a handprint, the coverage factor is the distance from the lowest part of the print, near the base of the palm, to the highest part of the print, at the fingertip. The scanner reports this distance in 100ths of an inch. Note that this is generally less than the height of the image, since white background can be present at both the top and bottom. A host application can compare this coverage factor for two handprints to determine whether they are sufficiently similar. Dissimilar handprints may indicate that there was a problem in capturing one or the other. Suggested Default Minimum Hand Coverage Factor = 400.

The roll time is the amount of time taken to roll the finger, in milliseconds. The capture process monitors finger motion. It starts capturing a roll as soon as the finger begins moving. It stops when

the finger is motionless for a few frames, or begins to roll in the opposite direction. The roll time represents the time from the first frame in the capture to the first frame in which the finger became motionless again, or began to roll back. Image quality may degrade in very fast rolls. Very slow rolls tend to stop the finger movement detection. A host application can use the roll time to require a particular pace from an operator performing a rolled capture. A roll time of 0 will indicate that the scanner has detected finger slippage. A typical roll time is on the order of 1500 mS. Suggested Default Minimum Roll Time = (Minimum Cover Factor * 10).

For a handprint, the roll time is the time taken when the cylinder was rotating, in milliseconds. The capture process tracks the amount of time taken for each line in the raw image and reports the sum. A host application can use this roll time, and the coverage factor (length of the palm and fingers), to decide whether the average time per inch rolled is within acceptable limits. Generally, this parameter is not used for handprints, since the sidebar speed indicators will provide the operator with feedback, and fast rolls will be rejected with status CSTATE_TOOFAST (see *Start Preview*).

All Captures:

The capture process divides the image into a lattice of small, rectangular cells. In each cell, it determines whether the finger made contact. For each cell in which it finds contact, it decides whether that cell is good, too light, or too dark.

The contact area is the ratio of cells in which the capture process found contact and the number of cells needed to cover the entire scanning surface, expressed as a percentage. The remaining members report a percentage of actual contact cells, not the total number of cells. The good area is the percentage of good cells within the contact cells. The dark area is the percentage of cells in which contact is too dark, the light area is the percentage of cells in which contact is too light, and the smeared area the percentage of cells smeared (low contrast). These measures can be used by the application, when the good_area falls below an acceptable threshold, to provide feedback to the operator on obtaining a better print. A high percentage of too light, dark, or smeared cells may indicate that the operator is not pressing hard enough, too hard, or moving the finger, respectively.

The image quality members will all be less than or equal to 100. However, since bad cells may be counted in more than one category, the sum of good, too dark, too light, and too smeared cells may be greater than 100. Suggested Defaults: Minimum Contact Area = none, Minimum Good Area = 75/50 (Finger/4-Slap or Palm).

This function will return information about the currently captured image. A host application may use this information, along with the decorated image, to provide the operator feedback on the quality of the captured image.

Coverage factor and roll time are valid only when PREVIEW_ROLL or PREVIEW_HAND capture type has been selected. The application-provided image callback function should retrieve this information during the CSTATE_CAPTURED callback, evaluate it, and reject or accept the image pending operator review via the callback return code (see *Start Preview*). **Calling this function at any other time will return an error.**

Error Codes:

TPAPI_UNINITIALIZED

Library has not been successfully initialized

TPAPI_PARAMETEROUTOFRANGE

hDev is an invalid handle, or *pParams* is a NULL pointer

TPAPI_INVALIDSTATE

Command cannot be completed in the current capture state

TPAPI_TIMEOUT

Timeout attempting to communicate with the device

TPAPI_BADRESPONSE

Unexpected or invalid response to device command

4.3.12. Query Switches

[C++] long STDCALL TP_QuerySwitches(U32 hDev, S32 Mode, U32 WaitMsec, U32 Prompt, U32 Freq[3], U32 Time[3], U16 *SwitchState)

[.NET] TpApiError QuerySwitches(eQrySwMode Mode, U32 WaitMsec, U32 Prompt, U32[] Freq, U32[] Time, ref U16 SwitchState)

Parameter:

hDev

Device handle

Mode

Mode of operation. Allowed values:

QRYSW_CLEAR	Clear prior switch states, prompt, and wait for a switch closure
QRYSW_NOCLEAR	Prompt and return when a switch closure is detected
QRYSW_NOWAIT	Return switch states immediately
QRYSW_CANCEL	Cancel previous query (i.e. wait)

WaitMsec

Wait timeout period in milliseconds. 0 = wait indefinitely

Prompt

0 = No audible prompt; 1-3 = Number of audible tones

Freq

Frequency of up to three audible tones (in Hertz)

Time

Duration of each of up to three audible tones (in milliseconds)

SwitchState¹

Bitmapped switch state

• Bit SW_BSCAN_LATCHED	Scan button pressed since last query
• Bit SW_BSAVE_LATCHED	Save button pressed since last query
• Bit SW_FSCAN_LATCHED	Scan foot-switch pressed since last query
• Bit SW_FSAVE_LATCHED	Save foot-switch pressed since last query
• Bit SW_NOT_BSCAN_NOW	Scan button currently NOT pressed
• Bit SW_NOT_BSAVE_NOW	Save button currently NOT pressed
• Bit SW_NOT_FSCAN_NOW	Scan foot-switch currently NOT pressed
• Bit SW_NOT_FSAVE_NOW	Save foot-switch currently NOT pressed
• Value SW_SWITCH_TIMEOUT	<i>WaitMsec</i> timeout expired

Check for button and foot-pedal switch closures on the device with an optional audible prompt.

Each switch has two items of state information. One tells whether an operator is currently pressing the switch, the other is a latched condition, which tells whether an operator has pressed a switch since the last time the switches were queried. If there are no attached foot-pedals they appear to be permanently open.

The time-out period begins after the scanner sounds the last tone in an audible prompt, if any. If an operator does not press a switch within this period, this command stops and reports the latest switch states.

Up to three audible tones can be used in succession as an audible prompt to an operator. A host application may specify any frequency, but the actual tone will be the nearest frequency of which a

¹ On all -ED devices, foot-pedal closures are returned as button closures.

scanner is capable. These are a finite set of tones in the range 37-9433 Hz. A frequency of zero means the speaker will be silent for the associated period. This allows a host application to sound a two-tone audible prompt with a pause between them.

Each tone from the array of *Freq[]* sounds for the associated *Time[]* period. Although each value is in milliseconds, the actual duration is the nearest 100-millisecond interval. If this is zero, then the tone will sound continuously until an operator presses a switch or the time-out period expires.

This function is NOT available during the capture process (see *StartPreview - final capture process states*) and will return TPAPI_BADRESPONSE if an earlier *Query Switches* command has not completed.

Error Codes:

TPAPI_UNINITIALIZED

Library has not been successfully initialized

TPAPI_PARAMETEROUTOFRANGE

hDev is an invalid handle, *Prompt* value invalid, or *SwitchState* is NULL

TPAPI_DEVICEUNAVAILABLE

Device not found on the bus

TPAPI_INVALIDSTATE

Command cannot be completed in the current capture state

TPAPI_TIMEOUT

Timeout attempting to communicate with the device

TPAPI_BADRESPONSE

Unexpected or invalid response to device command

4.3.13. Device Connected

[C++] BOOL STDCALL TP_DeviceConnected(U32 hDev)

[.NET] bool DeviceConnected()

Parameters:

hDev

Device handle

This function will return true if the device is detected on the bus. The bus is continuously monitored by the TPAPI library, which will return the current state of any connected devices.

4.3.14. Get Device Info

[C++] long STDCALL TP_GetDeviceInfo(U32 hDev, struct s_devInfo *devInfo)

[.NET] TpApiError GetDeviceInfo(ref s_devInfo devInfo)

Parameters:

hDev

Device handle

devInfo

struct s_devInfo

{

char assy_make [8]; // Assembly Make (e.g. "IDX")

char assy_model [16]; // Assembly Model (e.g. "TP4100")

char assy_sn [16]; // Assembly Serial Number (e.g. "123456")

char sw_version [12]; // Embedded Firmware Application Version (e.g. "1.01")

}

Buffer structure in which to return NULL-terminated device information strings

Returns the make, model, serial number, and firmware version of the device

Error Codes:

TPAPI_UNINITIALIZED

Library has not been successfully initialized

TPAPI_PARAMETEROUTOFRANGE

hDev is an invalid handle, or *devInfo* is NULL

TPAPI_DEVICEUNAVAILABLE

Device not found on the bus

TPAPI_TIMEOUT

Timeout attempting to communicate with the device

TPAPI_BADRESPONSE

Unexpected or invalid response to device command

4.3.15. Reset Device

[C++] long STDCALL TP_ResetDevice(U32 hDev)

[.NET] TpApiError ResetDevice()

Parameter:

hDev

Device handle

Initiates a software reset of the Live Scan hardware. This is function will block for approximately 10 seconds waiting to re-establish communications with the device.

Error Codes:

TPAPI_UNINITIALIZED

Library has not been successfully initialized

TPAPI_PARAMETEROUTOFRANGE

hDev is an invalid handle

TPAPI_DEVICEUNAVAILABLE

Device not found on the bus

TPAPI_TIMEOUT

Timeout attempting to communicate with the device

4.3.16. Is Calibration Supported

[C++] long STDCALL TP_IsCalibrationSupported(U32 hDev, enum eCaptureType capType, BOOL *result)

[.NET] TpApiError IsCalibrationSupported(eCaptureType capType, ref bool result)

Parameter:

hDev

Device handle

capType

See *Is Capture Type Supported*

result

Boolean value indicating calibration support

Returns an indication of support for calibration of the given capture.

Error Codes:

TPAPI_UNINITIALIZED

Library has not been successfully initialized

TPAPI_PARAMETEROUTOFRANGE

hDev is an invalid handle, or *result* is NULL

TPAPI_DEVICEUNAVAILABLE

Device not found on the bus

TPAPI_BADRESPONSE

Unexpected or invalid response to device command

TPAPI_TIMEOUT

Timeout attempting to communicate with the device

4.3.17. Is Calibrated

[C++] long STDCALL TP_IsCalibrated(U32 hDev, enum eCaptureType capType, BOOL *result)

[.NET] TpApiError IsCalibrated(eCaptureType capType, ref bool result)

Parameter:

hDev

Device handle

capType

See *Is Capture Type Supported*

result

Boolean value indicating calibration state

Detects if the equalization reference images have been correctly installed or the device has been successfully calibrated (see *Equalization Reference Images*).

Error Codes:

TPAPI_UNINITIALIZED

Library has not been successfully initialized

TPAPI_PARAMETEROUTOFRANGE

hDev is an invalid handle, or *result* is NULL

TPAPI_DEVICEUNAVAILABLE

Device not found on the bus

TPAPI_BADRESPONSE

Unexpected or invalid response to device command

TPAPI_TIMEOUT

Timeout attempting to communicate with the device

4.3.18. Calibrate Scanner

[C++] long STDCALL TP_CalibrateScanner(U32 hDev, enum eCaptureType capType)

[.NET] TpApiError CalibrateScanner(eCaptureType capType)

Parameter:

hDev

Device handle

capType

See *Is Capture Type Supported*

Calibrates the scanner associated with the capture type, and saves the resultant equalization reference image to the hard drive. The scanning surfaces should be thoroughly cleaned, free of streaks and lint, and the platen cover closed prior to performing a calibration.

Equalization is the process that maps the actual dynamic range from a scanner camera into the 8-bit grayscale range in all captured images. This mapping makes the background appear white and areas in contact with the scanning surface appear dark. It also compensates for the unique optical characteristics of each individual scanning unit (scanning surface, illumination, etc.).

Calibration should be performed occasionally to account for small variations in the optical characteristics of the system related to both age and usage.

Hand Scanner-Only:

The calibration process for a hand-scanning unit is more involved than for other types of units. The operator must rotate the drum in the same manner as when performing a hand capture, only without touching the scanning surface. An access cover opens to expose the encoder wheel so an operator can rotate the drum without touching the scanning surface. The scanner requires a minimum amount of rotation to complete each scan.

The scanner first sounds a high-pitched beep. This lets the operator know when to begin rotating the drum. The operator must continue this rotation until the scanner sounds another high-pitched beep, signaling successful completion of the calibration image capture process.

The scanner will sound a low-pitched beep to indicate an invalid scan, usually indicating that the drum was rolled too quickly or too slowly. When this happens, it terminates the calibration process, does not generate an equalization image, and returns a TPAPI_BADRESPONSE error code.

Error Codes:

TPAPI_UNINITIALIZED

Library has not been successfully initialized

TPAPI_PARAMETEROUTOFRANGE

hDev is an invalid handle, or *capType* not supported

TPAPI_DEVICEUNAVAILABLE

Device not found on the bus

TPAPI_BADRESPONSE

Unexpected or invalid response to device command

TPAPI_INVALIDSTATE

Command cannot be completed in the current state

TPAPI_TIMEOUT

Timeout attempting to communicate with the device

TPAPI_BADFILE

Error writing equalization reference image

4.3.19. Get / Set Resolution

[C++] long STDCALL TP_GetResolution (U32 hDev, enum eCaptureType capType, U32 *dpi)

[C++] long STDCALL TP_SetResolution (U32 hDev, enum eCaptureType capType, U32 dpi)

[.NET] TpApiError GetResolution (eCaptureType capType, ref U32 dpi)

[.NET] TpApiError SetResolution (eCaptureType capType, U32 dpi)

Parameters:

hDev

Device handle

capType

See *Is Capture Type Supported*

dpi

Resolution, in dots-per-inch, of the finished image; Valid values = 500 or 1000.

Gets or sets the resolution of the finished image of the given capture type *and all like base capture types*. Base capture types are CTYPE_ROLL, CTYPE_SLAP_ONE, CTYPE_SLAP_FOUR, CTYPE_SLAP_PALM, and CTYPE_HAND.

Error Codes:

TPAPI_UNINITIALIZED

Library has not been successfully initialized

TPAPI_PARAMETEROUTOFRANGE

hDev is an invalid handle, *capType* not supported, *dpi* a NULL reference, or value invalid or not supported

TPAPI_DEVICEUNAVAILABLE

Device not found on the bus

TPAPI_BADRESPONSE

Unexpected or invalid response to device command

TPAPI_INVALIDSTATE

Command cannot be completed in the current state

TPAPI_TIMEOUT

Timeout attempting to communicate with the device

4.3.20. Get / Set Max Hand Size

[C++] long STDCALL TP_GetMaxHandSize (U32 hDev, enum eHandSz *hndSz)

[C++] long STDCALL TP_SetMaxHandSize (U32 hDev, enum eHandSz hndSz)

[.NET] TpApiError GetMaxHandSize ref emaxHandSz hndSz)

[.NET] TpApiError SetMaxHandSize (emaxHandSz hndSz)

Parameters:

hDev

Device handle

hndSz

Maximum hand sizes:

TP_HAND_8_INCH 8 inches - FBI palm print card size (*default*)

TP_HAND_10_INCH 10 inches

Gets or sets the maximum length of a captured hand image for the device. The length of the finished image will be the actual size rolled, up to this maximum. This value should be set prior to retrieving the maximum video size (*Get Max Video Size*).

Error Codes:

TPAPI_UNINITIALIZED

Library has not been successfully initialized

TPAPI_PARAMETEROUTOFRANGE

hDev is an invalid handle, *hndSz* a NULL reference or value invalid or not supported

TPAPI_DEVICEUNAVAILABLE

Device not found on the bus

TPAPI_BADRESPONSE

The command is not supported on the attached device

TPAPI_INVALIDSTATE

Command cannot be completed in the current state

TPAPI_TIMEOUT

Timeout attempting to communicate with the device

4.3.21. Get / Set Slap Finger Thresholds

[C++] long STDCALL TP_GetSlapFngrThresholds(U32 hDev, struct s_fngrThlds *fngrThrld)

[C++] long STDCALL TP_SetSlapFngrThresholds(U32 hDev, struct s_fngrThlds *fngrThrld)

[.NET] TpApiError GetSlapFngrThresholds(ref s_fngrThlds fngThrld)

[.NET] TpApiError SetSlapFngrThresholds(s_fngrThlds fngThrld)

Parameter:

hDev

Device handle

fngThrld

Array of ten (10) structures (provided by the caller) storing contact thresholds for each finger

Get or Set the finger contact thresholds used by the device during a slap auto-capture operation. The first array entry (*fngThrld[0]*) is associated with the left little finger, the second the left ring finger, progressing left-to-right through the right little finger. Structure *s_fngThlds* contains three different 8-bit threshold levels: Good Contact, Okay Contact, Minimum Contact.

Error Codes:

TPAPI_UNINITIALIZED

Library has not been successfully initialized

TPAPI_PARAMETEROUTOFRANGE

hDev is an invalid handle, or *fngThrld* a NULL reference

TPAPI_DEVICEUNAVAILABLE

Device not found on the bus

TPAPI_BADRESPONSE

Unexpected or invalid response to device command

TPAPI_TIMEOUT

Timeout attempting to communicate with the device

4.3.22. Get / Set Slap Auto-Capture Timeouts

[C++] long STDCALL TP_GetSlapAutoCapTimeouts(U32 hDev, U32 *capDelay, U32 *capTmOut, U32 *altCapTmOut)

[C++] long STDCALL TP_SetSlapAutoCapTimeouts(U32 hDev, U32 capDelay, U32 capTmOut, U32 altCapTmOut)

[.NET] TpApiError GetSlapAutoCapTimeouts(ref U32 capDelay, ref U32 capTmOut, ref U32 altCapTmOut)

[.NET] TpApiError SetSlapAutoCapTimeouts(U32 capDelay, U32 capTmOut, U32 altCapTmOut)

Parameter:

hDev

Device handle

capDelay

Tenths-of-a-second delayed before triggering a capture after all fingers make "good" contact

capTmOut

Tenths-of-a-second before forcing a capture (regardless of contact thresholds) after finger contact is detected

altCapTmOut

Tenths-of-a-second delayed before triggering a capture after the number of fingers set in *Set Slap Auto-Capture Primary Fingers* have exceeded the "good" contact threshold.

Get or Set timeouts associated with an auto-capture operation.

Error Codes:**TPAPI_UNINITIALIZED**

Library has not been successfully initialized

TPAPI_PARAMETEROUTOFRANGE

hDev is an invalid handle, or Pointer parameters are NULL

TPAPI_DEVICEUNAVAILABLE

Device not found on the bus

TPAPI_BADRESPONSE

Unexpected or invalid response to device command

TPAPI_INVALIDSTATE

Command cannot be completed in the current state

TPAPI_TIMEOUT

Timeout attempting to communicate with the device

4.3.23. Get / Set Slap Auto-Capture Primary Fingers

[C++] long STDCALL TP_GetSlapAutoCapPrimaryFngers(U32 hDev, U32 *numFngers)

[C++] long STDCALL TP_SetSlapAutoCapPrimaryFngers (U32 hDev, U32 numFngers)

[.NET] TpApiError GetSlapAutoCapPrimaryFngers(ref U32 numFngers)

[.NET] TpApiError SetSlapAutoCapPrimaryFngers (U32 numFngers)

Parameter:**hDev**

Device handle

numFngers

Minimum number of fingers required to be detected during a slap auto-capture operation before initiating the alternate capture timer (see *Set Slap Auto-Capture Timeouts*).

Get or Set the number of fingers associated with the alternate auto-capture operation timeout. *A change in this parameter will not persist after a reboot of the device.*

Error Codes:**TPAPI_UNINITIALIZED**

Library has not been successfully initialized

TPAPI_PARAMETEROUTOFRANGE

hDev is an invalid handle, or *numFingers* a NULL reference

TPAPI_DEVICEUNAVAILABLE

Device not found on the bus

TPAPI_BADRESPONSE

Unexpected or invalid response to device command

TPAPI_INVALIDSTATE

Command cannot be completed in the current state

TPAPI_TIMEOUT

Timeout attempting to communicate with the device

4.3.24. Send Bulk Data

[C++] long STDCALL TP_SendBulkData(U32 hDev, enum eScnDataType dType, U8 *buffer, U32 bufLen)

[.NET] TpApiError SendBulkData(eScnDataType dType, ref byte[] buffer)

Parameter:

hDev

Device handle

dType

- SCN_DTYPE_DSP - Update DSP firmware. This should be followed by a *Reset Device* to activate the update.

buffer

Buffer containing data to transfer

bufLen

Number of bytes in *buffer*

Transfer the block of data to the attached device. This function is provided for maintenance and diagnostic use only.

Error Codes:

TPAPI_PARAMETEROUTOFRANGE

buffer is NULL or *bufLen* is 0

TPAPI_UNINITIALIZED

Library has not been successfully initialized

TPAPI_PARAMETEROUTOFRANGE

hDev is an invalid handle, *buffer* is NULL, or *bufLen* is 0

TPAPI_DEVICEUNAVAILABLE

Device not found on the bus

TPAPI_BADRESPONSE

Unexpected or invalid response to device command

TPAPI_INVALIDSTATE

Command cannot be completed in the current state

TPAPI_TIMEOUT

Timeout attempting to communicate with the device

5. TP-LSMULTI Supported Devices

This section lists the devices supported by the TouchPrint Live Scan Multi-Use SDK and their FBI certification names. This FBI certification name should be used in a NIST record to identify the certified device type, and on a hard copy fingerprint card in the location designated for the scanner.

Live Scan product name	FBI certification name
TP-3800HA	Identix TPFC-2
TP-3000A-ED	Identix TP-5700
TP-3100A-ED	Identix TP-5700
TP-3800A-ED	Identix TP-5700
TP-3800HA-ED	Identix TP-5700
TP-4x4A	Identix TP-4000
TP-4100A-ED	Identix TP-4100
TP-4100UA-ED	Identix TP-4100
Agile TP	Identix TP-4101
TP-4101B	Identix TP-4101
TP-4800PA-ED	Identix TP-4800
TP-5000A-ED	L-1 Identity Solutions TP-5750
TP-5100A-ED	L-1 Identity Solutions TP-5750
TP-5300A-ED	L-1 Identity Solutions TP-5300
TP-5000A-HD	L-1 Identity Solutions TP-5750
TP-5100A-HD	L-1 Identity Solutions TP-5750
TP-5300A-HD	L-1 Identity Solutions TP-5300

(Note: The TP-4x4A/TP-4000 is not FBI certified for roll captures.)

6. SDK File List

This section lists the files that are in the **TP-LSMULTI SDK** Installation package.

\ TP-LSMULTI SDK\Doc: Document files

TP-LSMULTI SDK Release Notes.pdf
TP-LSMULTI SDK Programmer's Manual.pdf (this document)

\ TP-LSMULTI SDK\bin: Drivers, Shared DLLs and Executable files

TPAPI.NetWrapper.dll	.NET wrapper assembly
tpapi.dll	Application interface to TouchPrint Live Scan device
saliJag1.dll	SALI director DLL
saliuni.dll	Low-level SALI Unibrain Interface DLL (<i>x86-only</i>)
saligen.dll	Low-level SALI Generic OHCI Interface DLL
saliusb.dll	Low-level SALI USB 2.0 Interface DLL
id1394.dll	TP-LSMULTI-SDK OHCI driver support library
id1394.sys	TP-LSMULTI-SDK OHCI driver
id1394.inf	TP-LSMULTI-SDK OHCI driver configuration file
idUsb.sys	TP-LSMULTI-SDK USB 2.0 driver
idUsb.inf	TP-LSMULTI-SDK USB 2.0 driver configuration file
IdImf.dll	Image Finishing library
IdImq.dll	Image Quality library
idxTiff.dll	TP-LSMULTI-SDK Tiff library
pthreadVC.dll	POSIX thread library
tpMultiDevice.exe	Sample MFC program demonstrating TP-LSMULTI API (<i>x86-only</i>)
TPAPI Sample App.exe	Sample MFC program demonstrating TP-LSMULTI SDK API
VBDemoApp.exe	Sample VB 6.0 program (<i>x86-only</i>)
TPAPI .NET Sample App.exe	Sample .NET program demonstrating the .NET Wrapper

\ TP-LSMULTI SDK\Tester: Maintenance Utility files

JagTest1.exe	TP-LSMULTI Maintenance Utility
JagTest1 Commands.pdf	Maintenance Utility Help
EquMod.exe	TP-3800HA Reference Image conversion utility (<i>x86 only</i>)

\ TP-LSMULTI SDK\Tester\Firmware: Firmware and FPGA Image files

Firmware Update Proc.txt	Firmware Update Procedure
FirmwareManifest.xml	Firmware Manifest XML file
FirmwareUpdate.exe	Sample .NET program demonstrating TP-LSMULTI firmware update
TP-5000.DSP.x.xx.idx	TP-3000A-ED Embedded Application Firmware file
TP-4000.DSP.x.xx.idx	TP-4xxxA Embedded Application Firmware file
TP-4100U.x.xx.idx	TP-4100UA Consolidated Firmware file
TP-4101.x.xx.idx	AgileTP/TP-4101B Consolidated Firmware file
TP-4800.x.xx.idx	TP-4800PA Consolidated Firmware file
TP-5300.x.xx.idx	TP-5300A Consolidated Firmware file
TP-5300ED.x.xx.idx	TP-5300A-ED Consolidated Firmware file
Jaguar.x.xx.idx	TP-3800HA Embedded Application Firmware file

\ TP-LSMULTI SDK\redist: Redistributables

TP_LSMULTI_MM_xx.msm	Merge Module
TP-LSMULTI MM Installation Guide.txt	

\ TP-LSMULTI SDK\Include: Shared Include files

tpapi.h	TPAPI function prototype definition file
memtiff.h	IdxTiff function prototype definition file
scnprtcl.h	TP-LSMULTI scanner protocol definition file

\ TP-LSMULTI SDK\Lib: Import libraries

tpapi.lib	Microsoft VC import library for linking to TPAPI.dll
IdxTiff.lib	Microsoft VC import library for linking to IdxTiff.dll

\ TP-LSMULTI SDK\src\TPAPI Sample App:**\ TP-LSMULTI SDK\src\TPAPI .NET Sample App:****\ TP-LSMULTI SDK\src\FirmwareUpdate:****\ TP-LSMULTI SDK\src\tpMultiDevice:****\ TP-LSMULTI SDK\src\VBDEmoApp:**

...	Source files for TPAPI MFC, .NET, and Visual Basic 6.0 <i>(32-bit only)</i> sample applications
-----	--

In addition, the following Microsoft redistributable support packages are installed:

MSVCRT	Microsoft C Runtime 6.0 <i>(x86-only)</i>
COMCAT	Microsoft Component Category Manager <i>(x86-only)</i>
OLEAUT32	Microsoft OLE 2.4 <i>(x86-only)</i>
MSVBVM60	Visual Basic Virtual Machine 6.0 <i>(x86-only)</i>
CRT	Visual C++ C-Runtime libraries
MFC	Visual C++ MFC libraries
dotnetfx2setup.exe	Microsoft .NET Framework 2.0 Redistributable

7. Driver Installation

The TP-LSMULTI SDK installer utilizes Microsoft's Driver Install Frameworks (DIFx) to install the USB 2.0 and IEEE1394 OHCI drivers provided in the installation package. Microsoft Logo Program-certified drivers require no further action. From time-to-time the drivers may be updated and released prior to being re-certified. In this case, the user may be required to direct the Windows XP driver installer to the location of the driver's INF file. The default location for this file is [INSTALLDIR]bin\idUsb or [INSTALLDIR]bin\id1394.

If the TP-LSMULTI SDK installer is not utilized during application deployment the application developer is responsible for installing or updating the necessary device drivers. See Windows Hardware Developer Central – Driver Installation (<http://www.microsoft.com/whdc/driver/install/default.mspx>) for more information.

7.1. USB 2.0 or IEEE 1394 OHCI Driver Update

To manually update the TP-LSMULTI-SDK USB or 1394 OHCI driver software perform the following steps:

1. Install the TP-LSMULTI SDK.
2. Connect and power on the scanner.

On a new installation, the Windows operating system will detect the new hardware, and may prompt for a driver INF file. To manually update the target PC with a new driver, bring up the properties of the driver in the Device Manager dialog, and select "Update Driver".

3. Browse to the [INSTALLDIR] bin\[idUsb|id1394] directory, and select *idusb.inf* for USB, *id1394.inf* for 1394 OHCI.
4. Accept all of the defaults.

8. Device Firmware Installation

If the SDK Release Notes indicate a change to one of the device firmware files, please update the firmware of the target device upon completion of the SDK file and driver install.

8.1. Firmware Update Utility

The device firmware can be updated manually by an operator using the Firmware Update utility provided with the SDK or a similar utility.

8.2. Application Firmware Verification & Update

Device firmware verification and update could also be embedded into an application's initialization process utilizing the *FirmwareManifest.xml* file included with the SDK. The *FirmwareManifest.xml* file provides the current SDK version along with the recommended device firmware versions and firmware filenames for each of the device models supported in a standard XML format. Refer to *TP-LSMULTI Supported Devices* to correlate Live Scan product names with FBI certification/model names.

Example *FirmwareManifest.xml* file format:

```
<?xml version="1.0"?>
<TP-LSMULTI-SDK version="7.10">
  <DeviceModels>
    <Model name="TP4000">
      <FirmwareVersion>7.15</FirmwareVersion>
      <FirmwareFile>TP-4000.DSP.7.15.idx</FirmwareFile>
    </Model>
    <Model name="TP5700">
      <FirmwareVersion>7.15</FirmwareVersion>
      <FirmwareFile>TP-5000.DSP.7.15.idx</FirmwareFile>
    </Model>
  </DeviceModels>
</TP-LSMULTI-SDK>
```

Utilizing the *Get Device Info assy_model* and *sw_version* information, the appropriate firmware file can be located, read from, and provided to the *Send Bulk Data* API function. On successful completion of the *Send Bulk Data* firmware update, the new firmware should be activated by resetting the device (see *Reset Device*).

9. Equalization Reference Images

Prior to initiating an image capture on a Live Scan device, an *equalization reference image* must be created for each of the capture types (single-finger, four-finger, hand). Applications should call the *CalibrateScanner* function at initialization time, or provide the operator a mechanism to manually calibrate and create reference images for the scanner.

Due to the complexity of the calibration procedure on a TP-3800HA, creating single and four-finger equalization reference images outside the factory is NOT SUPPORTED. Instead, these images are provided on the 3000 Series System Calibration Back-up Disk included with each unit. To install these images on the host PC, perform the following steps:

1. Cd [CDVOLUME]:\TP3800
2. Copy *.dbi [INSTALLPATH]\tester\
3. Copy *.txt [INSTALLPATH]\tester\
4. Cd [INSTALLPATH]\tester\
5. Run "equmod -d LWhite.dbi 66 720 1199 [CommonAppData]\slapRef.tif"
6. Run "equmod -d SWhite.dbi 66 320 749 [CommonAppData]\fingerRef.tif"
7. Erase *.dbi

The [CommonAppData] folder is "\Documents and Settings\All Users\Application Data" on XP and "\ProgramData" on Windows 7 systems. Hand scanner reference images should be created through the *CalibrateScanner* function.

Note: Equalization reference images are the result of a calibration process and are therefore unique for every scanner. These images must reside in the same directory as TPAPI.dll at run-time.

