USB LISM-PI26xx Interface

Cross-Platform Library Manual

Windows DLL and Linux Shared Library - Version 2.0

Legacy API and Extended Handle API

Use the *_ex functions for multi-device applications.

Coptonix GmbH
Falkentaler Steig 9
13467 Berlin, Germany
+49 30 617 412 48
support@coptonix.com
www.coptonix.com

Rev. 1.0 / July 2026

Contents

1 Introduction

This manual describes the USB LISM-PI26xx cross-platform host library for Windows and Linux, including x86, x86-64, ARM32 and AArch64 targets. The same library interface provides USB device discovery, buffered image acquisition, I2C access, Clock and Timing Generator control, persistent settings, status information and hardware identification.

Existing single-device applications can continue to use the legacy function names. New applications may also use the legacy API when only one interface is opened. Applications that open and operate more than one USB LISM-PI26xx interface at the same time must use the corresponding handle-based functions whose names end in *_ex.

Scope: Register bit definitions, sensor-specific timing limits and electrical characteristics are documented in the LISM-PI26xx data sheet. This library manual repeats only the register information that is required to use the host API correctly.

Scope of the API

- USB enumeration and access to descriptor information.
- Legacy single-device operation and handle-based multi-device operation.
- Per-device host ring buffers, background USB bulk-read threads and runtime acquisition-thread priority control.
- Internal and external I2C master transactions.
- Acquisition, trigger, timing and analog front-end configuration.
- EEPROM settings, initialization status and hardware identification.

API Naming

Legacy functions use the historical names such as ls_opendevicebyindex, ls_getpipe and ls_setstate. They operate on one current legacy device context. The related *_ex functions accept an explicit ls_handle_t as their first parameter and operate only on that device context.

The suffix _ext identifies access to the external I2C channel. It is independent of the *_ex suffix. For example, ls_readi2c_ext_ex is the handle-based version of the external-I2C read function.

2 Supported Platforms, Library Files and Bindings

Library Files

Platform	Application link/load name	USB backend	Process type
Windows 32-bit	usblismpi2632.dll	WinUSB	32-bit application
Windows 64-bit	usblismpi2664.dll	WinUSB	64-bit application
Linux 32-bit / ARM32	libusblismpi2632.so	libusb-1.0	32-bit process
Linux 64-bit / x86-64 / AArch64	libusblismpi2664.so	libusb-1.0	64-bit process

Windows applications must load the DLL that matches the bitness of the application process. A 64-bit operating system can run both variants, but a 32-bit application cannot load the 64-bit DLL and a 64-bit application cannot load the 32-bit DLL.

Linux applications should link against the unversioned link name. A library release is installed as four related files so applications do not need to encode the complete release number in their link settings:

Linux file	Purpose
libusblismpi26xx.so.<major>.<minor>.<release>	Real library file containing the complete major, minor and release version.
libusblismpi26xx.so.<major>.<minor>	Minor-version symbolic link.
libusblismpi26xx.so.<major>	SONAME/major-version symbolic link.
libusblismpi26xx.so	Unversioned development and application link name.

Important: In the names above, 26xx is replaced by 2632 or 2664 according to the process architecture. <major> is the ABI major version used by the SONAME link.

Headers and Language Bindings

The current public integration files are located in the include directory of the library package. The portable C/C++ header is the primary declaration reference. The language bindings provide equivalent declarations or wrappers and expose the public LS_ERR_* values and portable acquisition-thread priority constants.

File	Purpose
C_Cpp/usblismpi26.h	Portable C/C++ declarations for Windows and Linux/ARM. Contains the legacy API, the handle-based *_ex multi-device API, public error codes and portable acquisition-thread priority constants.
Cpp_Windows_DynamicLoader/usblismpi26_loader.h	Windows runtime-loader declarations, usable from C and C++ with Microsoft Visual C++ or Embarcadero C++Builder.
Cpp_Windows_DynamicLoader/usblismpi26_loader.c	C implementation using LoadLibrary/GetProcAddress. Add this file and the matching header to the Windows application project; no import library is required.
Qt/usblismpi26.h	Qt-specific C++ declarations using Qt fixed-width integer types and automatic 32/64-bit library-name selection.
Delphi_Lazarus/usblismpi26.pas	Pascal import unit for Delphi and Lazarus on Windows and Linux. Uses implicit dynamic linking.
Python/usblismpi26.py	ctypes binding with a legacy single-device class and a handle-based device class for *_ex operation.
CSharp/usblismpi26.cs	C#/.NET binding with platform-aware native-library loading.
VB/usblismpi26.vb	VB.NET binding with platform-aware native-library loading.

C and C++ Linking

On Windows, include C_Cpp/usblismpi26.h and link against the import library supplied for the matching DLL. When no import library is available or runtime DLL selection is preferred, use the two files in Cpp_Windows_DynamicLoader instead.

On Linux, include C_Cpp/usblismpi26.h and link against libusblismpi2632.so or libusblismpi2664.so. The runtime loader must be able to locate the corresponding SONAME link and real library file through the system library path, an application rpath or LD_LIBRARY_PATH.

All exported functions use a C ABI. LS_CALL expands to __stdcall on Windows and to the default C calling convention on Linux. The header is enclosed in extern "C" when included from C++. The common ls_result_t name is uint32_t on Windows and int32_t on Linux, preserving positive native Windows errors and negative Linux/libusb errors.

Linux Requirements

The Linux shared library uses libusb-1.0. Install the runtime package on the target system and the development package when compiling or linking an application:

Debian/Ubuntu example

```
sudo apt-get install libusb-1.0-0 libusb-1.0-0-dev
```

Non-root USB access requires a udev rule for vendor ID 19D1 and product ID 0011. The exact group name and access mode should follow the security policy of the target system.

Example udev rule

```
SUBSYSTEM=="usb", ATTR{idVendor}=="19d1", ATTR{idProduct}=="0011", MODE="0660", GROUP="plugdev"
```

After installing or changing the rule, reload the rules and reconnect the USB interface or trigger udev. The application user must be a member of the selected group.

3 API Concepts

Legacy Single-Device API

The legacy API owns one current device context. The application first stores the host FIFO and packet configuration with ls_initialize, enumerates the connected interfaces and opens one device by index or serial number. All later legacy data-transfer, I2C and CTG functions operate on that current device.

Opening another device through a legacy open function first closes the current legacy device. The legacy API is therefore not suitable for simultaneous multi-device acquisition, but it remains appropriate for existing software and new applications that operate exactly one device at a time.

Extended Handle API and Multi-Device Operation

Each successful *_ex open function returns an opaque ls_handle_t. The handle identifies one independent USB connection, host ring buffer, acquisition thread, acquisition-thread priority, packet length and endpoint timeout. Pass that handle to every subsequent *_ex function for the same device.

Several handles may be open concurrently. Do not mix legacy device-specific calls with a handle that was opened through *_ex. For multi-device applications, use *_ex consistently for opening, data transfer, I2C/CTG access and closing.

Important: An ls_handle_t is opaque. The application must not dereference it, copy internal data from it or reuse it after ls_closedevice_ex has completed.

Complete USB control and I2C operations are serialized internally for each device context, including multi-step register reads. Calls that use different handles remain independent and may run concurrently. Internal serialization prevents transaction interleaving, but the application remains responsible for the intended order of higher-level operations.

Device Index, Serial Number and Handle

Identifier	Meaning	Lifetime
Device index	Zero-based position in the list created by the latest ls_enumdevices call.	Valid only for the current enumeration list.
Serial number	Persistent USB descriptor string assigned to the physical interface.	Stable for the physical device.
Device handle	Opaque context returned by ls_opendevicebyindex_ex or ls_opendevicebyserial_ex.	Valid until the corresponding ls_closedevice_ex call.

A new enumeration can change device indexes. Existing open *_ex handles continue to identify their device contexts, but the application should not reuse old index assignments after rebuilding the list. Serial-number based opening is recommended when a device must keep a fixed application role.

Host FIFO, Packet Length and Endpoint Timeout

Each open device has a host-side ring buffer. The background acquisition thread repeatedly reads one packet_length block from the USB bulk-IN endpoint and writes the received bytes to that ring buffer. The effective pipe_size must be at least as large as the effective packet_length. Application threads retrieve data with ls_getpipe or ls_getpipe_ex.

Value	Meaning
pipe_size	Capacity of the host ring buffer in bytes. A value of 0 selects the 1 MiB default. Every positive value is used unchanged on Windows and Linux. The effective value must be at least packet_length.
packet_length	Requested bytes per USB bulk read. It must match one complete device transfer block or an integer multiple. A value of 0 in an *_ex open function queries the current value from the device.
endpoint timeout	Maximum wait for one USB bulk-IN attempt. The value must be greater than zero; the default is 100 ms. A timeout ends only the current read attempt and is independent of I2C/CTG control-transfer timeouts and application-side FIFO waits.

The endpoint timeout limits one USB bulk-read attempt only and must be greater than zero. If a read times out with no data, the acquisition thread immediately starts the next read. If the transfer returns partial data together with a timeout, the received bytes are preserved and written to the host FIFO. The endpoint timeout may therefore be shorter than the interval between complete data blocks.

I2C Address Format and Byte Order

The API uses 8-bit I2C addresses. The factory-default LISM-PI26xx address is 0xFE. The least significant bit of a stored module address must be 0. Do not convert the value to a 7-bit address before passing it to the library.

Typed 16-bit and 32-bit I2C helper functions transfer the most-significant byte first. Raw I2C functions do not change the byte order of the caller-provided buffer.

Timeout and Return-Value Rules

- Most status-returning functions return 0 on success and a non-zero error value on failure. The historical Linux legacy ls_getpipe is an exception: it returns a negative error code or the positive number of bytes read.
- A timeout value of 0 performs an immediate/non-blocking check only for the pipe wait functions.
- An endpoint timeout of 0 is invalid and returns LS_ERR_INVALID_TIMEOUT; this rule does not change the separate pipe-wait semantics.
- A pipe wait timeout of 0xFFFFFFFF waits indefinitely until the requested condition is met or the device is closed.
- I2C and CTG timeout values are passed to the USB control transfers used for the command.
- Use ls_geterrorstring to convert a returned error value to text. Numeric error values are platform-specific.

Pointers returned by ls_getvendorname, ls_getproductname, ls_getserialnumber and ls_geterrorstring reference separate library-managed per-thread buffers. A pointer remains valid until the next call to the same string-returning function on the same application thread. Copy the string when it must remain valid beyond that point.

4 Basic Workflows

Legacy Single-Device Workflow

C/C++ example

```
ls_result_t result;
uint32_t bytes_read = 0;
uint8_t buffer[4096];

#ifdef _WIN32
ls_initialize(4u * 1024u * 1024u, 4096u, 0u, 0, "LISMApp");
#else
ls_initialize(4 * 1024 * 1024, 4096);
#endif
if (ls_enumdevices() > 0) {
    result = ls_opendevicebyindex(0);
    if (result == LS_RET_OK) {
#ifdef _WIN32
        result = ls_setstate(LS_LISM_ADDR, 1);
#else
        result = ls_setstate(LS_LISM_ADDR, 1, 1000);
#endif
        if (result == LS_RET_OK)
            result = ls_waitforpipecount(4096, (int32_t*)&bytes_read, 1000);
        if (result == LS_RET_OK) {
#ifdef _WIN32
            result = ls_getpipe(buffer, 4096, &bytes_read);
#else
            int32_t read_result = ls_getpipe(buffer, 4096);
            if (read_result < 0)
                result = read_result;
            else {
                bytes_read = (uint32_t)read_result;
                result = LS_RET_OK;
            }
#endif
        }
#ifdef _WIN32
        ls_setstate(LS_LISM_ADDR, 0);
#else
        ls_setstate(LS_LISM_ADDR, 0, 1000);
#endif
        ls_closedevice();
    }
}
ls_shutdown();
```

The exact legacy signatures differ between Windows and Linux. Use the current public header or the language-specific binding instead of copying declarations from this example.

Multi-Device Workflow with *_ex

C/C++ example
```
ls_handle_t camera_a = NULL;
ls_handle_t camera_b = NULL;
ls_result_t result;

if (ls_enumdevices() >= 2) {
    result = ls_opendevicebyindex_ex(0, 4u * 1024u * 1024u, 0, &camera_a);
    if (result == LS_RET_OK)
        result = ls_opendevicebyindex_ex(1, 4u * 1024u * 1024u, 0, &camera_b);

    if (camera_a != NULL)
        ls_setstate_ex(camera_a, LS_LISM_ADDR, 1, 1000);
    if (camera_b != NULL)
        ls_setstate_ex(camera_b, LS_LISM_ADDR, 1, 1000);

    /* Read and process each handle independently. */

    if (camera_b != NULL)
        ls_closedevice_ex(camera_b);
    if (camera_a != NULL)
        ls_closedevice_ex(camera_a);
}
ls_shutdown();
```

Important: If the second open fails, close the first handle before leaving the error path. ls_shutdown can be used during application shutdown as a final safety measure to close every remaining legacy and *_ex device context.

Reading a Complete Data Block with a Bounded Wait

C/C++ example
```
uint8_t buffer[4096];
uint32_t bytes_read = 0;
int32_t available = 0;

result = ls_waitforpipecount_ex(handle, 4096, &available, 1000);
if (result == LS_RET_OK) {
    result = ls_getpipe_ex(handle, buffer, 4096, &bytes_read);
    if (result == LS_RET_OK && bytes_read == 4096) {
        /* Process one complete 4096-byte block. */
    }
}
```

The copy mode of ls_getpipe_ex waits for the exact requested byte count and has no timeout parameter. Use ls_waitforpipecount_ex first when the application requires a bounded wait.

5 Function Reference

The declarations in this section are taken from the public C/C++ header in include/C_Cpp/usblismpi26.h. The extended declaration is listed first when a related legacy declaration also exists.

The language-specific bindings expose the same operations using the normal types and conventions of the target language. Refer to the binding source for exact class or method names.

5.1 Library and Device Management

Library Version - ls_libversion

Returns the public library API version as a 16-bit value. The high byte contains the major version and the low byte contains the minor version. For example, 0x0200 represents version 2.0.

The function does not require initialization, enumeration or an open device and can be called immediately after the library has been loaded.

Legacy API - Windows DLL
```
uint16_t LS_CALL ls_libversion(void);
```

```
Legacy API - Linux shared library
uint16_t ls_libversion(void);
```

Return value. The encoded library version. A value of 0x0200 represents version 2.0.

Initialize Legacy Configuration - ls_initialize

Stores the host FIFO size and packet length that will be used by later legacy open calls. It does not enumerate devices, open USB communication or start an acquisition thread.

The Windows legacy form also accepts the historical thread-priority and registered-message parameters. The Linux legacy form accepts only the FIFO and packet settings. The *_ex API does not use ls_initialize because its open functions receive the FIFO and packet settings directly.

```
Legacy API - Windows DLL
uint32_t LS_CALL ls_initialize(
    uint32_t pipe_size,
    uint32_t packet_length,
    uint32_t thread_class,
    int32_t thread_priority,
    const char* message_id
);

Legacy API - Linux shared library
void ls_initialize(int32_t pipe_size, int32_t packet_length);
```

Parameters

Parameter	Direction	Description
pipe_size	In	Requested host ring-buffer size in bytes. Value 0 selects 1 MiB on Windows and Linux. Every positive value is used unchanged. The effective value must be at least packet_length.
packet_length	In	USB bulk-read size in bytes. Value 0 selects the 4096-byte legacy default. A non-positive Linux value also selects that default.
thread_class	In	Windows legacy process-priority-class parameter retained for compatibility.
thread_priority	In	Windows thread-priority value applied to the legacy acquisition thread when possible.
message_id	In	Windows null-terminated string passed to RegisterWindowMessageA.

Return value. Windows returns the registered Windows message ID, or 0 when no ID is registered. The Linux function has no return value.

Enumerate Devices - ls_enumdevices / ls_devicecount

ls_enumdevices scans the USB bus for interfaces with the USB LISM-PI26xx vendor and product identifiers and rebuilds the internal indexed device list. If a detected interface is already open through *_ex, its device and string descriptors are read again through the existing USB handle. Interfaces that are not open are accessed temporarily for descriptor reading.

If a legacy device is open, ls_enumdevices closes it before rebuilding the list. Existing *_ex handles remain open and continue to identify the same physical devices. Because the indexed list is rebuilt, device indexes can change after every enumeration; do not reuse old index assignments. ls_devicecount returns the number of entries in the current list without performing another scan.

```
Legacy API - Windows DLL
int32_t LS_CALL ls_enumdevices(void);

uint8_t LS_CALL ls_devicecount(void);

Legacy API - Linux shared library
int32_t ls_enumdevices(void);

uint8_t ls_devicecount(void);
```

Return value. ls_enumdevices returns the number of detected devices, or -1 if the USB backend could not be initialized. ls_devicecount returns the current list size as an 8-bit value.

Device Descriptor Information

These functions return information stored during the most recent enumeration. The index is a zero-based position in the current device list. An open device is not required for these descriptor queries.

The firmware version is encoded as a 16-bit major/minor value. The manufacturer, product and serial-number functions return pointers to separate library-managed per-thread buffers. Each pointer remains valid until the next call to the same descriptor function on that thread.

Legacy API - Windows DLL
```
uint16_t LS_CALL ls_getfwversion(int32_t index);

const char* LS_CALL ls_getvendorname(int32_t index);

const char* LS_CALL ls_getproductname(int32_t index);

const char* LS_CALL ls_getserialnumber(int32_t index);
```

Legacy API - Linux shared library
```
uint16_t ls_getfwversion(int32_t index);

const char* ls_getvendorname(int32_t index);

const char* ls_getproductname(int32_t index);

const char* ls_getserialnumber(int32_t index);
```

Parameters

Parameter	Direction	Description
index	In	Zero-based device-list index created by the latest ls_enumdevices call.

Return value. ls_getfwversion returns the encoded version or 0 for an invalid index. String functions return an empty string for an invalid index.

Current Legacy Device Index - ls_currentdeviceindex

Returns the enumeration index associated with the currently open legacy device. This function reports only the legacy device context and does not describe any handles opened through *_ex.

Legacy API - Windows DLL
```
int32_t LS_CALL ls_currentdeviceindex(void);
```

Legacy API - Linux shared library
```
int32_t ls_currentdeviceindex(void);
```

Return value. The zero-based index of the current legacy device, or -1 when no legacy device is open.

Open Device by Index

Opens the device identified by an index from the current enumeration list. The library creates the host ring buffer, opens the USB interface and starts the background bulk-read thread.

The legacy form uses the settings stored by ls_initialize and replaces the current legacy device. The *_ex form creates an independent device context and returns it through handle. A physical USB interface cannot be opened twice in the same process, whether through the legacy API or another handle. Duplicate-open protection is atomic across concurrent open attempts.

Extended API - Windows and Linux
```
ls_result_t LS_CALL ls_opendevicebyindex_ex(
    int32_t index,
    uint32_t pipe_size,
    uint32_t packet_length,
    ls_handle_t* handle
);
```

Legacy API - Windows DLL
```
uint32_t LS_CALL ls_opendevicebyindex(int32_t index);
```

Legacy API - Linux shared library
```
int32_t ls_opendevicebyindex(int32_t index);
```

Parameters

Parameter	Direction	Description
index	In	Zero-based index from the latest enumeration list.
pipe_size	In	Host ring-buffer size in bytes. Value 0 selects the 1 MiB default.
packet_length	In	Bytes per USB bulk read. Value 0 queries the current packet length from the device before starting the read thread.
handle	Out	Receives the new opaque device handle on success; set to NULL on failure.

Return value. 0 on success; otherwise a device-list, duplicate-open, USB, packet-length, FIFO-size, allocation or thread-start error.

Behavior

- Call ls_enumdevices before opening by index.
- The background read thread starts immediately after a successful open, even if acquisition is currently stopped.
- When packet_length is 0, the *_ex open function queries the current value from the device and fails if the query fails or returns zero. Opening also fails with LS_ERR_INVALID_DATALEN when the effective pipe_size is smaller than the effective packet_length.

Open Device by Serial Number

Searches the current enumeration list for an exact serial-number match and opens that physical interface. Serial-number based opening is useful when the application must assign fixed roles independently of USB enumeration order.

The legacy and *_ex behavior after selection is identical to the corresponding index-based open functions.

Extended API - Windows and Linux
```
ls_result_t LS_CALL ls_opendevicebyserial_ex(
    const char* serial_num,
    uint32_t pipe_size,
    uint32_t packet_length,
    ls_handle_t* handle
);
```

Legacy API - Windows DLL
```
uint32_t LS_CALL ls_opendevicebyserial(const char* serial_num);
```

Legacy API - Linux shared library
```
int32_t ls_opendevicebyserial(const char* serial_num);
```

Parameters

Parameter	Direction	Description
serial_num	In	Null-terminated serial-number string to locate in the current enumeration list.
pipe_size	In	Host ring-buffer size for the new *_ex context. Value 0 selects 1 MiB.
packet_length	In	USB bulk-read size. Value 0 queries the value from the device.
handle	Out	Receives the new opaque device handle on success.

Return value. 0 on success; otherwise an enumeration, serial-number, duplicate-open, USB or allocation error.

Close Device

Stops the background read thread, cancels blocked host-FIFO operations, closes the USB interface and releases the device context.

ls_closedevice closes the current legacy device and returns success when no legacy device is open. ls_closedevice_ex requires a valid open handle; the handle becomes invalid after a successful close and must not be reused.

Extended API - Windows and Linux
```
ls_result_t LS_CALL ls_closedevice_ex(ls_handle_t handle);
```

Legacy API - Windows DLL
```
uint32_t LS_CALL ls_closedevice(void);
```

Legacy API - Linux shared library
```
int32_t ls_closedevice(void);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle returned by an *_ex open function.

Return value. 0 on success. The extended function returns LS_ERR_INVALID_HANDLE for NULL, arbitrary or already-closed handles. CloseDevice can return LS_ERR_TIMEOUT if active calls cannot be drained within the internal wait period, LS_ERR_DEVICE_ACTIVE when another close is already in progress, or a native backend error if USB shutdown fails.

Behavior

- Threads blocked in ls_getpipe*, ls_waitforpipe* or ls_waitforpipecount* are released during close.
- Pipe waits or reads released by an intentional close return LS_ERR_PIPE_CANCELLED. A physical USB disconnect is reported with the original native backend disconnect error instead.

Close All Devices - ls_shutdown

Closes the current legacy device and every remaining handle-based device context owned by the process. It then finalizes and releases the USB backend and clears the enumerated device list. This function is intended for orderly application shutdown and for cleanup after an error path that may have left more than one handle open.

Legacy API - Windows DLL
```
uint32_t LS_CALL ls_shutdown(void);
```

Legacy API - Linux shared library
```
int32_t ls_shutdown(void);
```

Return value. 0 when all devices and backend resources were released successfully; otherwise the first close error encountered. A later ls_enumdevices call recreates and initializes the backend normally.

Error Text - ls_geterrorstring

Converts a public library error or a native USB-backend error value to a readable null-terminated string. On Windows, unmapped Win32, WinUSB and SetupAPI errors are formatted using the operating-system error text. On Linux, native negative libusb errors and the private USB LISM-PI26xx error range are identified separately.

The Linux public API does not return positive POSIX errno values. The returned pointer references the per-thread error-string buffer, remains valid until the next ls_geterrorstring call on the same thread and must not be modified or freed.

Legacy API - Windows DLL
```
const char* LS_CALL ls_geterrorstring(uint32_t error_code);
```

Legacy API - Linux shared library
```
const char* ls_geterrorstring(int32_t error_code);
```

Parameters

Parameter	Direction	Description
error_code	In	32-bit error value returned by a library function.

Return value. Pointer to a null-terminated ANSI string. Passing 0 returns SUCCESS.

5.2 USB Configuration and Data Transfer

Endpoint Timeout - Set / Get

Controls the timeout used by the background USB bulk-IN read operation. The value must be greater than zero and limits one endpoint read attempt, not the complete acquisition session.

The legacy setter changes the value used by the next legacy open and must be called while no legacy device is open. The extended setter applies the value directly to the specified open handle. The default value is 100 ms. After a read timeout, the acquisition thread immediately starts another read; bytes received before a timeout are retained in the host FIFO.

Extended API - Windows and Linux
```
ls_result_t LS_CALL ls_seteptimeout_ex(ls_handle_t handle, uint32_t timeout_ms);
```

```
ls_result_t LS_CALL ls_geteptimeout_ex(ls_handle_t handle, uint32_t* timeout_ms);

Legacy API - Windows DLL
uint32_t LS_CALL ls_seteptimeout(uint32_t timeout_ms);

uint32_t LS_CALL ls_geteptimeout(void);

Legacy API - Linux shared library
int32_t ls_seteptimeout(uint32_t timeout_ms);

uint32_t ls_geteptimeout(void);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended functions.
timeout_ms	In/Out	Endpoint timeout in milliseconds. Set values must be greater than zero; the getter writes the current value to the supplied pointer.

Return value. Setters return 0 on success. A value of 0 returns LS_ERR_INVALID_TIMEOUT; other failures can report an invalid handle or a native backend error. The legacy getter returns the timeout directly. The extended getter returns a result code and writes the current value through its output pointer.

Important: Endpoint timeout 0 is invalid. A timeout ends only the current USB read attempt; the acquisition thread immediately retries. Partial data received before a timeout is retained, so the endpoint timeout may be shorter than the camera trigger interval.

Acquisition Thread Priority - Set / Get

Sets or reads the portable priority class of the background acquisition thread belonging to an open handle. These functions are available only in the handle-based *_ex API because each open handle owns its own acquisition thread.

The priority can be changed while the device is open and while acquisition is running; no device reopen or thread restart is required. The portable classes are mapped to suitable native thread priorities by the library. The historical Windows thread-priority argument of ls_initialize remains limited to legacy opens.

```
Extended API - Windows and Linux
ls_result_t LS_CALL ls_setthreadpriority_ex(ls_handle_t handle, int32_t priority);
ls_result_t LS_CALL ls_getthreadpriority_ex(ls_handle_t handle, int32_t* priority);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle whose background acquisition thread is selected.
priority	In/Out	Portable priority constant supplied to the setter or receiving the current value from the getter.

Supported priority values

Constant	Meaning
LS_THREAD_PRIORITY_NORMAL (0)	Normal acquisition-thread priority.
LS_THREAD_PRIORITY_ABOVE_NORMAL (1)	Priority above the normal application-thread class.
LS_THREAD_PRIORITY_HIGH (2)	High acquisition-thread priority for demanding continuous transfers.

Return value. 0 on success. Invalid or closed handles return LS_ERR_INVALID_HANDLE. Unsupported values return LS_ERR_INVALID_THREAD_PRIORITY. If the operating system or runtime cannot apply the requested class, the setter returns LS_ERR_THREAD_PRIORITY_FAILED.

Important: Higher priority can improve scheduling of the USB acquisition thread but can also reduce CPU time available to processing and user-interface threads. Use the lowest class that provides reliable transfer behavior.

Packet Length - Set / Get

The packet length is the byte count requested by each background USB bulk read. It must match the complete block size produced by the device or an integer multiple of that block size. The effective host FIFO size must be at least as large as the packet length.

ls_setpacketlength changes only the stored legacy configuration and therefore requires the legacy device to be closed. It rejects zero and also rejects a value larger than the configured legacy FIFO size. There is no separate *_ex setter because packet_length is supplied when an extended device is opened. The getter functions query the current packet length reported by the USB interface.

```
Extended API - Windows and Linux
ls_result_t LS_CALL ls_getpacketlength_ex(
    ls_handle_t handle,
    uint32_t* packet_length,
    uint32_t timeout_ms
);

Legacy API - Windows DLL
uint32_t LS_CALL ls_setpacketlength(uint32_t packet_length);

uint32_t LS_CALL ls_getpacketlength(uint32_t* packet_length);

Legacy API - Linux shared library
int32_t ls_setpacketlength(int32_t packet_length);

int32_t ls_getpacketlength(int32_t* packet_length, uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
packet_length	In/Out	Legacy setter value or output variable receiving the device-reported packet length.
handle	In	Open device handle for ls_getpacketlength_ex.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy query.

Return value. 0 on success; otherwise an invalid-handle, invalid-packet-length, invalid-data-length, timeout or control-transfer error.

Behavior

- A zero packet length returns LS_ERR_INVALID_PACKETLENGTH; a value larger than the configured legacy FIFO returns LS_ERR_INVALID_DATALEN.
- For 16-bit pixel data, one image line normally contains pixel_count x 2 bytes.
- If CFG1 buffers several lines per transfer, multiply the line size by the configured image count.

Custom Firmware Value - ls_customfirmware / ls_customfirmware_ex

Reads the 32-bit custom-firmware value reported by the USB interface. The value can be used for diagnostics and compatibility decisions; its interpretation is defined by the installed USB-interface firmware.

The Windows legacy function uses the default control-transfer timeout. Linux legacy and the extended function accept an explicit timeout.

```
Extended API - Windows and Linux
ls_result_t LS_CALL ls_customfirmware_ex(
    ls_handle_t handle,
    uint32_t* custom_firmware,
    uint32_t timeout_ms
);

Legacy API - Windows DLL
uint32_t LS_CALL ls_customfirmware(uint32_t* custom_firmware);

Legacy API - Linux shared library
int32_t ls_customfirmware(uint32_t* custom_firmware, uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for ls_customfirmware_ex.
custom_firmware	Out	Receives the 32-bit custom-firmware value reported by the USB interface.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy functions.

Return value. 0 on success; otherwise an invalid-handle, timeout, response-length or control-transfer error.

USB Configuration Register CFG1 - Set / Get

Writes or reads the complete bit-coded USB interface CFG1 byte. CFG1 selects whether counter metadata replaces the final two pixel values of each line, selects the counter format and defines how many complete image lines the device groups into one USB transfer.

The host packet length must match the resulting complete transfer size: packet_length = pixel_count x 2 x image_count. The image count is decoded from bits [6:2] and is independent of the number of lines per frame.

Extended API - Windows and Linux

```
ls_result_t LS_CALL ls_setcfg1_ex(
    ls_handle_t handle,
    uint8_t cfg1,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_getcfg1_ex(
    ls_handle_t handle,
    uint8_t* cfg1,
    uint32_t timeout_ms
);
```

Legacy API - Windows DLL

```
uint32_t LS_CALL ls_setcfg1(uint8_t cfg1);

uint32_t LS_CALL ls_getcfg1(uint8_t* cfg1);
```

Legacy API - Linux shared library

```
int32_t ls_setcfg1(uint8_t cfg1, uint32_t timeout_ms);

int32_t ls_getcfg1(uint8_t* cfg1, uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended functions.
cfg1	In/Out	Complete CFG1 byte. Set reserved bit 7 to zero and combine the required fields described below.
timeout_ms	In	Control-transfer timeout for the extended and Linux legacy functions.

Return value. 0 on success; otherwise an invalid-handle, timeout, short-response or control-transfer error.

Bits	Values	Function	Detailed behavior
0	0 / 1	Counter append enable	0: pixel data is transferred without counter metadata. 1: four bytes of counter metadata replace the last two 16-bit pixel values of every transferred line.
1	0 / 1	Counter format	Used only when bit 0 = 1. 0: one monotonically increasing 32-bit global line counter. 1: a 16-bit line number within the frame followed by a 16-bit frame counter.
6:2	0...31	Buffered line count minus one	image_count = ((cfg1 >> 2) & 0x1F) + 1. The device starts one USB transfer after grouping 1...32 complete lines, subject to its transfer-buffer capacity.
7	0	Reserved	Must be written as zero and ignored when read.

Typical maximum grouping values are 2 lines at 4096 pixels, 4 at 2048 pixels, 8 at 1024 pixels, 16 at 512 pixels and 32 at 256 pixels. Select a smaller value when required by the application or host-buffer design.

Important: CFG1 changes become active after a power cycle. Query the device packet length after applying a new CFG1 value and configure the host packet length accordingly.

Wait for Any Pipe Data

Blocks until at least one byte is present in the selected host FIFO, the timeout expires or the device is closed. The function does not remove data from the FIFO.

Use this function when any amount of newly available data is sufficient. Use ls_waitforpipecount* when a complete line or other fixed-size block is required.

Extended API - Windows and Linux

```
ls_result_t LS_CALL ls_waitforpipe_ex(ls_handle_t handle, uint32_t timeout_ms);
```

Legacy API - Windows DLL

```
uint32_t LS_CALL ls_waitforpipe(uint32_t timeout_ms);
```

Legacy API - Linux shared library

```
int32_t ls_waitforpipe(uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended function.
timeout_ms	In	0 = immediate check; 0xFFFFFFFF = wait indefinitely; otherwise maximum wait in milliseconds.

Return value. 0 when data is available. Otherwise returns LS_ERR_TIMEOUT, LS_ERR_INVALID_HANDLE, LS_ERR_PIPE_CANCELLED during intentional close, or a native backend error such as a physical-disconnect result.

Wait for a Pipe Byte Count

Blocks until at least count bytes are present in the selected host FIFO. The current number of available bytes is returned through available. No data is removed.

This function is the preferred way to wait for a complete line or transfer block before calling ls_getpipe*. It provides a bounded timeout while ls_getpipe* copy mode itself waits indefinitely for the exact requested byte count.

Extended API - Windows and Linux
```
ls_result_t LS_CALL ls_waitforpipecount_ex(
    ls_handle_t handle,
    int32_t count,
    int32_t* available,
    uint32_t timeout_ms
);
```

Legacy API - Windows DLL
```
uint32_t LS_CALL ls_waitforpipecount(
    int32_t count,
    int32_t* available,
    uint32_t timeout_ms
);
```

Legacy API - Linux shared library
```
int32_t ls_waitforpipecount(int32_t count, int32_t* available, uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended function.
count	In	Minimum required byte count. A negative value is invalid. A value of 0 performs an immediate fill-level query.
available	Out	Receives the current FIFO fill level. It is set to 0 when a close/cancel condition releases the wait.
timeout_ms	In	0 = immediate check; 0xFFFFFFFF = wait indefinitely; otherwise maximum wait in milliseconds.

Return value. 0 when at least count bytes are available. Otherwise returns LS_ERR_TIMEOUT, LS_ERR_INVALID_DATALEN, LS_ERR_INVALID_HANDLE, LS_ERR_PIPE_CANCELLED during intentional close, or a native backend error.

Read or Query the Host Pipe

Provides the application access to raw image bytes stored in the selected host FIFO. In copy mode, every variant waits until exactly to_read bytes are available, copies that complete byte count to buffer and removes the bytes from the FIFO. No variant performs a partial copy.

All variants provide query mode when buffer == NULL or to_read == 0. No data is removed. Windows legacy and ls_getpipe_ex write the current FIFO fill level to read_count; Linux legacy returns the fill level directly as its non-negative return value.

Extended API - Windows and Linux
```
ls_result_t LS_CALL ls_getpipe_ex(
    ls_handle_t handle,
    void* buffer,
    uint32_t to_read,
    uint32_t* read_count
);
```

Legacy API - Windows DLL
```
uint32_t LS_CALL ls_getpipe(void* buffer, uint32_t to_read, uint32_t* read_count);
```

Legacy API - Linux shared library
```
int32_t ls_getpipe(void* buffer, uint32_t to_read);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended function.
buffer	Out	Destination buffer. NULL selects non-copy query mode for every variant.
to_read	In	Exact number of bytes to copy. Zero selects query mode for every variant. The value must not exceed the host FIFO capacity.
read_count	Out	Receives bytes copied, or the current FIFO fill level in query mode. This parameter is not present in Linux legacy because its return value carries the copied or queried byte count.

Return value. Windows legacy and ls_getpipe_ex return 0 on success and report the copied byte count or queried FIFO fill level through read_count. Linux legacy returns the exact positive byte count after a copy, the current FIFO fill level after a query, or a negative error code on failure. A successful query returns 0 when the FIFO is empty. Possible failures include LS_ERR_INVALID_DATALEN, LS_ERR_INVALID_HANDLE, LS_ERR_PIPE_CANCELLED during intentional close, or a native backend error.

Behavior

- Copy mode has no timeout and waits until the complete requested byte count is available or the device is closed.
- Use ls_waitforpipecount* first when the application requires a bounded wait.
- The application must provide a destination buffer at least to_read bytes long.

Measured Data Rate - ls_getfps / ls_getfps_ex

Returns the measured number of image-data bytes written to the host FIFO per second. The historical function name contains FPS, but the returned unit is bytes per second, not frames per second.

The acquisition thread updates the value approximately once per second. Divide by the bytes per line to estimate the line rate. For a 2048-pixel line without additional transfer grouping, the line size is 4096 bytes.

Extended API - Windows and Linux
```
uint32_t LS_CALL ls_getfps_ex(ls_handle_t handle);
```

Legacy API - Windows DLL
```
uint32_t LS_CALL ls_getfps(void);
```

Legacy API - Linux shared library
```
uint32_t ls_getfps(void);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for ls_getfps_ex.

Return value. Measured bytes per second. Returns 0 when the selected device is invalid, closed or has not transferred data during the measurement interval.

Host and Hardware FIFO Behavior

The host ring buffer is independent of the LISM-PI26xx hardware FIFO. Stopping acquisition with ls_setstate(..., 0) stops new acquisition commands but does not clear either FIFO and does not cancel an application thread waiting in ls_getpipe*. Remaining host-side data can still be drained after acquisition is stopped.

If a device is closed while acquisition is active, or if it is closed immediately after stopping before complete transfer blocks have been drained, residual data can remain in the hardware. After the device is opened again, the new background read thread can receive that old data immediately. Applications that require a clean acquisition boundary should stop acquisition and drain or deliberately discard all complete data blocks that are still available.

When CFG1 groups more than one line into each USB transfer, an incomplete line group can remain in the USB-interface hardware buffer and cannot be read or drained by the host. For example, with image_count = 4, stopping after only one buffered line leaves an incomplete group. After acquisition is started again, new lines complete that group, so the first received four-line transfer can contain both residual and new lines. To establish a clean acquisition boundary, read and discard the first complete CFG1 transfer block after restarting, then begin processing with the following block.

Important: If the application does not read the host FIFO quickly enough, the library producer waits for free host-buffer space. The device-side FIFO can then fill, and the device determines which later data is lost. Select a sufficiently large pipe_size and process data continuously.

5.3 I2C Bus Interface

The USB interface acts as I2C master. Internal functions route transactions to the LISM-PI26xx channel, while function names containing _ext route transactions to the external user I2C channel. Raw transfers accept lengths from 1 to 64 bytes. The default module address is the 8-bit value 0xFE.

I2C Multiplexer Channel - Set / Get

Selects or reports the I2C multiplexer channel used by manual bus routing. Channel 0 is connected to the LISM-PI26xx module and channel 1 is connected to the external user bus.

Most high-level library functions select the required route through their internal or external request type. Direct multiplexer control is mainly useful for diagnostics and specialized low-level access.

Extended API - Windows and Linux

```
ls_result_t LS_CALL ls_setmuxchannel_ex(
    ls_handle_t handle,
    uint8_t mux,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_getmuxchannel_ex(
    ls_handle_t handle,
    int8_t* mux,
    uint32_t timeout_ms
);
```

Legacy API - Windows DLL

```
uint32_t LS_CALL ls_setmuxchannel(uint8_t mux);

uint32_t LS_CALL ls_getmuxchannel(int8_t* mux);
```

Legacy API - Linux shared library

```
int32_t ls_setmuxchannel(uint8_t mux, uint32_t timeout_ms);

int32_t ls_getmuxchannel(int8_t* mux, uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended functions.
mux	In/Out	0 = internal LISM-PI26xx channel; 1 = external user channel. The getter can return -1 when no valid channel is active.
timeout_ms	In	Control-transfer timeout for the extended and Linux legacy functions.

Return value. 0 on success; otherwise an invalid-handle, timeout, I2C status or multiplexer-disabled error.

Raw Internal I2C Read / Write

Transfers a caller-defined byte sequence on the internal LISM-PI26xx I2C channel. The library does not interpret register addresses or data fields in the supplied buffer.

For a raw read, length is both input and output: initialize it with the requested byte capacity, and after success it contains the number of bytes actually returned. Internal raw transactions retry a temporary I2C NACK before reporting failure.

Extended API - Windows and Linux

```
ls_result_t LS_CALL ls_readi2c_ex(
    ls_handle_t handle,
    uint8_t i2c_address,
    void* buffer,
    uint16_t* length,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_writei2c_ex(
    ls_handle_t handle,
    uint8_t i2c_address,
    const void* buffer,
    uint16_t length,
    uint32_t timeout_ms
);
```

Legacy API - Windows DLL

```
uint32_t LS_CALL ls_readi2c(uint8_t i2c_address, void* buffer, uint16_t* length);
```

```
uint32_t LS_CALL ls_writei2c(uint8_t i2c_address, const void* buffer, uint16_t length);

Legacy API - Linux shared library
int32_t ls_readi2c(
    uint8_t i2c_address,
    void* buffer,
    uint16_t* length,
    uint32_t timeout_ms
);

int32_t ls_writei2c(
    uint8_t i2c_address,
    const void* buffer,
    uint16_t length,
    uint32_t timeout_ms
);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended functions.
i2c_address	In	8-bit target address, normally 0xFE for the LISM-PI26xx module.
buffer	In/Out	Write source or read destination buffer.
length	In/Out	Transfer length from 1 to 64 bytes. For reads, input is requested capacity and output is transferred bytes.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy functions.

Return value. 0 on success; otherwise an invalid-length, invalid-handle, timeout, NACK, I2C-status or control-transfer error.

Raw External I2C Read / Write

Transfers a caller-defined byte sequence to an I2C device connected to the external user channel. The function routes the operation to the external bus without using the internal LISM-PI26xx register helpers.

The caller is responsible for the protocol expected by the external device, including register-address bytes, byte order, write-cycle delays and any required repeated transaction sequence.

```
Extended API - Windows and Linux
ls_result_t LS_CALL ls_readi2c_ext_ex(
    ls_handle_t handle,
    uint8_t i2c_address,
    void* buffer,
    uint16_t* length,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_writei2c_ext_ex(
    ls_handle_t handle,
    uint8_t i2c_address,
    const void* buffer,
    uint16_t length,
    uint32_t timeout_ms
);

Legacy API - Windows DLL
uint32_t LS_CALL ls_readi2c_ext(uint8_t i2c_address, void* buffer, uint16_t* length);

uint32_t LS_CALL ls_writei2c_ext(uint8_t i2c_address, const void* buffer, uint16_t length);

Legacy API - Linux shared library
int32_t ls_readi2c_ext(
    uint8_t i2c_address,
    void* buffer,
    uint16_t* length,
    uint32_t timeout_ms
);

int32_t ls_writei2c_ext(
    uint8_t i2c_address,
    const void* buffer,
    uint16_t length,
    uint32_t timeout_ms
);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended functions.
i2c_address	In	8-bit address of the external I2C target.
buffer	In/Out	Write source or read destination buffer.
length	In/Out	Transfer length from 1 to 64 bytes. Read length is input/output.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy functions.

Return value. 0 on success; otherwise an invalid-length, invalid-handle, timeout, NACK, I2C-status or control-transfer error.

Read 1-, 2- or 4-Byte Register

Writes the one-byte register address to the internal I2C target and then reads the requested value size. The helper verifies that the complete response length was received before changing the caller output value.

Two-byte and four-byte values are assembled in big-endian order, with the most-significant byte received first.

Extended API - Windows and Linux

```
ls_result_t LS_CALL ls_i2cread1byte_ex(
    ls_handle_t handle,
    uint8_t i2c_address,
    uint8_t i2c_register,
    uint8_t* value,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_i2cread2bytes_ex(
    ls_handle_t handle,
    uint8_t i2c_address,
    uint8_t i2c_register,
    uint16_t* value,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_i2cread4bytes_ex(
    ls_handle_t handle,
    uint8_t i2c_address,
    uint8_t i2c_register,
    uint32_t* value,
    uint32_t timeout_ms
);
```

Legacy API - Windows DLL

```
uint32_t LS_CALL ls_i2cread1byte(
    uint8_t i2c_address,
    uint8_t i2c_register,
    uint8_t* value
);

uint32_t LS_CALL ls_i2cread2bytes(
    uint8_t i2c_address,
    uint8_t i2c_register,
    uint16_t* value
);

uint32_t LS_CALL ls_i2cread4bytes(
    uint8_t i2c_address,
    uint8_t i2c_register,
    uint32_t* value
);
```

Legacy API - Linux shared library

```
int32_t ls_i2cread1byte(
    uint8_t i2c_address,
    uint8_t i2c_register,
    uint8_t* value,
    uint32_t timeout_ms
);

int32_t ls_i2cread2bytes(
    uint8_t i2c_address,
    uint8_t i2c_register,
    uint16_t* value,
    uint32_t timeout_ms
);
```

```
int32_t ls_i2cread4bytes(
    uint8_t i2c_address,
    uint8_t i2c_register,
    uint32_t* value,
    uint32_t timeout_ms
);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended functions.
i2c_address	In	8-bit address of the internal I2C target.
i2c_register	In	One-byte register/command address written before the read.
value	Out	Receives an 8-bit, 16-bit or 32-bit value according to the selected function.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy functions.

Return value. 0 on success; otherwise a write-phase, read-phase, response-length, NACK, timeout or handle error.

Write 1-, 2- or 4-Byte Register

Writes a one-byte register address followed by the supplied value to the internal I2C channel. The typed helpers avoid manual byte splitting and use the byte order expected by the LISM-PI26xx register protocol.

Two-byte and four-byte values are transmitted in big-endian order, most-significant byte first.

Extended API - Windows and Linux

```
ls_result_t LS_CALL ls_i2cwrite1byte_ex(
    ls_handle_t handle,
    uint8_t i2c_address,
    uint8_t i2c_register,
    uint8_t value,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_i2cwrite2bytes_ex(
    ls_handle_t handle,
    uint8_t i2c_address,
    uint8_t i2c_register,
    uint16_t value,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_i2cwrite4bytes_ex(
    ls_handle_t handle,
    uint8_t i2c_address,
    uint8_t i2c_register,
    uint32_t value,
    uint32_t timeout_ms
);
```

Legacy API - Windows DLL

```
uint32_t LS_CALL ls_i2cwrite1byte(
    uint8_t i2c_address,
    uint8_t i2c_register,
    uint8_t value
);

uint32_t LS_CALL ls_i2cwrite2bytes(
    uint8_t i2c_address,
    uint8_t i2c_register,
    uint16_t value
);

uint32_t LS_CALL ls_i2cwrite4bytes(
    uint8_t i2c_address,
    uint8_t i2c_register,
    uint32_t value
);
```

Legacy API - Linux shared library

```
int32_t ls_i2cwrite1byte(
    uint8_t i2c_address,
    uint8_t i2c_register,
    uint8_t value,
    uint32_t timeout_ms
);
```

```
int32_t ls_i2cwrite2bytes(
    uint8_t i2c_address,
    uint8_t i2c_register,
    uint16_t value,
    uint32_t timeout_ms
);

int32_t ls_i2cwrite4bytes(
    uint8_t i2c_address,
    uint8_t i2c_register,
    uint32_t value,
    uint32_t timeout_ms
);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended functions.
i2c_address	In	8-bit address of the internal I2C target.
i2c_register	In	One-byte register address.
value	In	8-bit, 16-bit or 32-bit value according to the selected function.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy functions.

Return value. 0 on success; otherwise an invalid-handle, timeout, NACK, I2C-status or control-transfer error.

Write I2C Command Without Data

Writes only the one-byte command/register address to the internal I2C target. No data byte follows. Use this helper for command registers such as UPDATE_PARAMS, STORE_SETTINGS, RELOAD_SETTINGS, RESET_SETTINGS and HW_RESET.

Extended API - Windows and Linux

```
ls_result_t LS_CALL ls_i2cwritecmd_ex(
    ls_handle_t handle,
    uint8_t i2c_address,
    uint8_t i2c_command,
    uint32_t timeout_ms
);
```

Legacy API - Windows DLL

```
uint32_t LS_CALL ls_i2cwritecmd(uint8_t i2c_address, uint8_t i2c_command);
```

Legacy API - Linux shared library

```
int32_t ls_i2cwritecmd(uint8_t i2c_address, uint8_t i2c_command, uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended function.
i2c_address	In	8-bit address of the internal I2C target.
i2c_command	In	One-byte command/register value.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy functions.

Return value. 0 on success; otherwise an invalid-handle, timeout, NACK, I2C-status or control-transfer error.

LISM-PI26xx I2C Address - Set / Get / Save

ls_setslaveaddress writes a proposed new 8-bit module address to the temporary address register. ls_getslaveaddress reads that temporary value back for verification. ls_saveslaveaddress commits the temporary address to EEPROM.

The module continues to respond at its current address until it is restarted. The saved address is used after the next power cycle. Address storage is separate from the general ls_savesettings command.

Extended API - Windows and Linux

```
ls_result_t LS_CALL ls_setslaveaddress_ex(
    ls_handle_t handle,
    uint8_t address,
    uint8_t new_address,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_getslaveaddress_ex(
    ls_handle_t handle,
    uint8_t address,
    uint8_t* new_address,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_saveslaveaddress_ex(
    ls_handle_t handle,
    uint8_t address,
    uint32_t timeout_ms
);
```

Legacy API - Windows DLL

```
uint32_t LS_CALL ls_setslaveaddress(uint8_t address, uint8_t new_address);

uint32_t LS_CALL ls_getslaveaddress(uint8_t address, uint8_t* new_address);

uint32_t LS_CALL ls_saveslaveaddress(uint8_t address);
```

Legacy API - Linux shared library

```
int32_t ls_setslaveaddress(uint8_t address, uint8_t new_address, uint32_t timeout_ms);

int32_t ls_getslaveaddress(uint8_t address, uint8_t* new_address, uint32_t timeout_ms);

int32_t ls_saveslaveaddress(uint8_t address, uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended functions.
address	In	Current 8-bit module address used to send the command.
new_address	In/Out	Proposed 8-bit address. Its least-significant bit must be 0.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy functions.

Return value. 0 on success; otherwise an I2C, timeout, invalid-handle or control-transfer error.

Important: Save the temporary address with ls_saveslaveaddress and then power-cycle the module. Until the restart, continue to use the old address.

5.4 Clock and Timing Generator

The CTG functions are typed wrappers around LISM-PI26xx I2C registers. address normally contains 0xFE. The extended functions additionally accept a device handle and an explicit control-transfer timeout. Register values and timing limits must be selected for the installed sensor type.

Clock and Timing Generator Hardware Reset

Sends the command-only HW_RESET register value 0xFF to the LISM-PI26xx module. The command resets the Clock and Timing Generator and its volatile operating state without closing the USB interface.

Saved settings are reapplied by the module initialization sequence. The host application should verify initialization status before restarting acquisition.

Extended API - Windows and Linux

```
ls_result_t LS_CALL ls_hw_reset_ex(
    ls_handle_t handle,
    uint8_t address,
    uint32_t timeout_ms
);
```

Legacy API - Windows DLL

```
uint32_t LS_CALL ls_hw_reset(uint8_t address);
```

Legacy API - Linux shared library

```
int32_t ls_hw_reset(uint8_t address, uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended function.
address	In	8-bit LISM-PI26xx I2C address, normally 0xFE.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy functions.

Return value. 0 when the reset command was transmitted successfully; otherwise an I2C, timeout or handle error.

Resume / Suspend Clock and Timing Generation

ls_resume writes 0xAA to the RESUME_GENERATOR register and enables Clock and Timing Generator operation. ls_suspend writes 0x00 and stops timing-signal generation while keeping I2C access available.

Suspension preserves the configured register values. Resume restores timing generation using the existing configuration.

Extended API - Windows and Linux

```
ls_result_t LS_CALL ls_resume_ex(
    ls_handle_t handle,
    uint8_t address,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_suspend_ex(
    ls_handle_t handle,
    uint8_t address,
    uint32_t timeout_ms
);
```

Legacy API - Windows DLL

```
uint32_t LS_CALL ls_resume(uint8_t address);

uint32_t LS_CALL ls_suspend(uint8_t address);
```

Legacy API - Linux shared library

```
int32_t ls_resume(uint8_t address, uint32_t timeout_ms);

int32_t ls_suspend(uint8_t address, uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended functions.

Parameter	Direction	Description
address	In	8-bit LISM-PI26xx I2C address.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy functions.

Return value. 0 on success; otherwise an I2C, timeout or handle error.

Important: Stop acquisition before suspending the generator so the acquisition state and timing outputs are changed in a controlled sequence.

Apply Updated Parameters - ls_updateparam

Sends the command-only UPDATE_PARAMS command (register 0x01). The communication microcontroller then reads the current operating parameter values from the Clock and Timing Generator and updates its I2C-accessible register copies with those values.

This is a CTG-to-microcontroller synchronization and readback operation. It does not apply microcontroller register values to the CTG, does not reset the CTG and does not store settings in EEPROM. After the command completes, the getter functions return the refreshed parameter values reported by the CTG.

Extended API - Windows and Linux
```
ls_result_t LS_CALL ls_updateparam_ex(
    ls_handle_t handle,
    uint8_t address,
    uint32_t timeout_ms
);
```

Legacy API - Windows DLL
```
uint32_t LS_CALL ls_updateparam(uint8_t address);
```

Legacy API - Linux shared library
```
int32_t ls_updateparam(uint8_t address, uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended function.
address	In	8-bit LISM-PI26xx I2C address.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy functions.

Return value. 0 on success; otherwise an I2C, timeout or handle error.

Acquisition State - Set / Get

Writes or reads the START_ACQUISITION register. A state value of 1 starts acquisition according to the configured mode and timing. A state value of 0 stops generation of new acquisition cycles.

Stopping acquisition affects only the device acquisition state. It does not clear the host FIFO, cancel a blocked pipe read or guarantee that the device hardware FIFO is already empty.

Extended API - Windows and Linux
```
ls_result_t LS_CALL ls_setstate_ex(
    ls_handle_t handle,
    uint8_t address,
    uint8_t state,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_getstate_ex(
    ls_handle_t handle,
    uint8_t address,
    uint8_t* state,
    uint32_t timeout_ms
);
```

Legacy API - Windows DLL
```
uint32_t LS_CALL ls_setstate(uint8_t address, uint8_t state);

uint32_t LS_CALL ls_getstate(uint8_t address, uint8_t* state);
```

Legacy API - Linux shared library
```
int32_t ls_setstate(uint8_t address, uint8_t state, uint32_t timeout_ms);
```

```
int32_t ls_getstate(uint8_t address, uint8_t* state, uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended functions.
address	In	8-bit LISM-PI26xx I2C address.
state	In/Out	0 = acquisition stopped; 1 = acquisition started.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy functions.

Return value. 0 on success; otherwise an I2C, timeout or handle error.

Important: After writing state 0, drain all required residual data before closing or beginning an acquisition that must start with an empty data stream.

Acquisition Mode Configuration - Set / Get

Writes or reads the complete bit-coded MODE_CONFIG register. Bits [2:0] select the acquisition trigger mode. Bits 4 and 5 configure the quadrature-encoder trigger and are used only when the mode field selects quadrature operation.

The mode parameter is the complete register byte, not only the mode number. Construct it by combining the required fields and keep all reserved bits at zero.

Extended API - Windows and Linux

```
ls_result_t LS_CALL ls_setmodeconfig_ex(
    ls_handle_t handle,
    uint8_t address,
    uint8_t mode,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_getmodeconfig_ex(
    ls_handle_t handle,
    uint8_t address,
    uint8_t* mode,
    uint32_t timeout_ms
);
```

Legacy API - Windows DLL

```
uint32_t LS_CALL ls_setmodeconfig(uint8_t address, uint8_t mode);

uint32_t LS_CALL ls_getmodeconfig(uint8_t address, uint8_t* mode);
```

Legacy API - Linux shared library

```
int32_t ls_setmodeconfig(uint8_t address, uint8_t mode, uint32_t timeout_ms);

int32_t ls_getmodeconfig(uint8_t address, uint8_t* mode, uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended functions.
address	In	8-bit LISM-PI26xx I2C address.
mode	In/Out	Complete MODE_CONFIG byte containing the trigger mode and optional encoder control bits.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy functions.

Return value. 0 on success; otherwise an I2C, timeout or handle error.

Bits	Value	Meaning	Detailed operation
2:0	0	Free running	While acquisition is enabled, the module continuously integrates and outputs lines without an external trigger.
2:0	1	External rising edge - single line	Each rising edge at the external trigger input starts one line acquisition.
2:0	2	External high level	Acquisition runs while the external trigger input remains high. The pulse width determines how many lines are acquired.
2:0	3	Internal software trigger	The CTG generates periodic trigger events. The interval is defined by SOFT_TRIGGER_PERIOD.
2:0	4	Quadrature encoder	Encoder direction is selected by bit 4. Bit 5 determines whether every valid count or the programmable QUAD_COUNT value triggers a line.

Bits	Value	Meaning	Detailed operation
2:0	5	External rising edge - multi-line	Each rising edge starts one frame containing the number of lines configured by LINES_PER_FRAME.
2:0	6 or 7	Reserved	Not defined for application use. Current CTG firmware falls back to free-running behavior; applications must not select these values.
3	0	Reserved	Must be zero.
4	0 / 1	Encoder direction	Quadrature mode only. 0: up-counting / clockwise direction. 1: down-counting / counterclockwise direction.
5	0 / 1	Encoder count enable	Quadrature mode only. 0: every valid encoder count can trigger one line. 1: trigger after the number of events configured by QUAD_COUNT.
7:6	0	Reserved	Must be written as zero and ignored when read.

Data Interface Configuration - Set / Get

Writes or reads the complete bit-coded DATA_IF_CONFIG register. Bits [2:0] select the data-clock and pixel-clock frequencies. Bits 3 through 6 select the active edge or active level of the data clock and synchronization outputs.

The clock selection also determines the duration represented by one START_PULSE_HIGH or START_PULSE_LOW count. Recalculate both start-pulse values whenever bits [2:0] are changed.

Extended API - Windows and Linux

```
ls_result_t LS_CALL ls_setifconfig_ex(
    ls_handle_t handle,
    uint8_t address,
    uint8_t config,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_getifconfig_ex(
    ls_handle_t handle,
    uint8_t address,
    uint8_t* config,
    uint32_t timeout_ms
);
```

Legacy API - Windows DLL

```
uint32_t LS_CALL ls_setifconfig(uint8_t address, uint8_t config);

uint32_t LS_CALL ls_getifconfig(uint8_t address, uint8_t* config);
```

Legacy API - Linux shared library

```
int32_t ls_setifconfig(uint8_t address, uint8_t config, uint32_t timeout_ms);

int32_t ls_getifconfig(uint8_t address, uint8_t* config, uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended functions.
address	In	8-bit LISM-PI26xx I2C address.
config	In/Out	Complete DATA_IF_CONFIG byte containing clock selection and output polarity fields.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy functions.

Return value. 0 on success; otherwise an I2C, timeout or handle error.

Bits	Value	Selected setting	Effect
2:0	0	Data clock 400 kHz; pixel clock 200 kHz	One START-pulse count = 5 us.
2:0	1	Data clock 500 kHz; pixel clock 250 kHz	One START-pulse count = 4 us.
2:0	2	Data clock 800 kHz; pixel clock 400 kHz	One START-pulse count = 2.5 us.
2:0	3	Data clock 1 MHz; pixel clock 500 kHz	One START-pulse count = 2 us.
2:0	4	Data clock 2 MHz; pixel clock 1 MHz	One START-pulse count = 1 us.
2:0	5	Data clock 4 MHz; pixel clock 2 MHz	One START-pulse count = 500 ns.
2:0	6	Data clock 10 MHz; pixel clock 5 MHz	One START-pulse count = 200 ns.
2:0	7	Data clock 20 MHz; pixel clock 10 MHz	One START-pulse count = 100 ns.
3	0 / 1	Data-clock polarity	0: pixel data changes on the falling edge; the receiver samples on the rising edge. 1: data changes on the rising edge; the receiver samples on the falling edge.
4	0 / 1	LINE_VALID polarity	0: active high. 1: active low.
5	0 / 1	FRAME_VALID polarity	0: active high. 1: active low.
6	0 / 1	Trigger-output polarity	0: active high. 1: active low.
7	0	Reserved	Must be written as zero and ignored when read.

Sensor Start Pulse High / Low - Set / Get

Sets or reads the high and low count values of the sensor start pulse. START_PULSE_HIGH primarily defines integration time, while START_PULSE_LOW completes the line period and provides time for sensor readout.

The combined ls_setstpulse and ls_getstpulse functions transfer both adjacent 32-bit registers in one operation. The actual time represented by each count depends on the pixel-clock selection in DATA_IF_CONFIG.

Extended API - Windows and Linux

```
ls_result_t LS_CALL ls_setsthigh_ex(
    ls_handle_t handle,
    uint8_t address,
    uint32_t high_value,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_setstlow_ex(
    ls_handle_t handle,
    uint8_t address,
    uint32_t low_value,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_setstpulse_ex(
    ls_handle_t handle,
    uint8_t address,
    uint32_t high_value,
    uint32_t low_value,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_getsthigh_ex(
    ls_handle_t handle,
    uint8_t address,
    uint32_t* high_value,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_getstlow_ex(
    ls_handle_t handle,
    uint8_t address,
    uint32_t* low_value,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_getstpulse_ex(
    ls_handle_t handle,
    uint8_t address,
    uint32_t* high_value,
    uint32_t* low_value,
    uint32_t timeout_ms
);
```

Legacy API - Windows DLL

```
uint32_t LS_CALL ls_setsthigh(uint8_t address, uint32_t high_value);

uint32_t LS_CALL ls_setstlow(uint8_t address, uint32_t low_value);

uint32_t LS_CALL ls_setstpulse(uint8_t address, uint32_t high_value, uint32_t low_value);

uint32_t LS_CALL ls_getsthigh(uint8_t address, uint32_t* high_value);

uint32_t LS_CALL ls_getstlow(uint8_t address, uint32_t* low_value);

uint32_t LS_CALL ls_getstpulse(uint8_t address, uint32_t* high_value, uint32_t* low_value);
```

Legacy API - Linux shared library

```
int32_t ls_setsthigh(uint8_t address, uint32_t high_value, uint32_t timeout_ms);

int32_t ls_setstlow(uint8_t address, uint32_t low_value, uint32_t timeout_ms);

int32_t ls_setstpulse(uint8_t address, uint32_t high_value, uint32_t low_value, uint32_t timeout_ms);

int32_t ls_getsthigh(uint8_t address, uint32_t* high_value, uint32_t timeout_ms);

int32_t ls_getstlow(uint8_t address, uint32_t* low_value, uint32_t timeout_ms);

int32_t ls_getstpulse(uint8_t address, uint32_t* high_value, uint32_t* low_value, uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended functions.
address	In	8-bit LISM-PI26xx I2C address.
high_value	In/Out	32-bit START_PULSE_HIGH count.
low_value	In/Out	32-bit START_PULSE_LOW count.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy functions.

Return value. 0 on success; otherwise an I2C, timeout, response-length or handle error.

Pixel clock	Time per count
200 kHz	5 us
250 kHz	4 us
400 kHz	2.5 us
500 kHz	2 us
1 MHz	1 us
2 MHz	500 ns
5 MHz	200 ns
10 MHz	100 ns

Important: The valid minimum high/low counts and the relationship between START pulse timing and integration time depend on the installed line sensor. Use the sensor-specific timing information.

Lines per Frame - Set / Get

Writes or reads the 16-bit LINES_PER_FRAME register. The value defines how many image lines are grouped while FRAME_VALID remains active and is also used by the multi-line external trigger mode.

LINE_VALID identifies each individual line. FRAME_VALID starts before the first line and ends after the configured final line according to the module timing.

Extended API - Windows and Linux

```
ls_result_t LS_CALL ls_setlinesperframe_ex(
    ls_handle_t handle,
    uint8_t address,
    uint16_t lines,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_getlinesperframe_ex(
    ls_handle_t handle,
    uint8_t address,
    uint16_t* lines,
    uint32_t timeout_ms
);
```

Legacy API - Windows DLL

```
uint32_t LS_CALL ls_setlinesperframe(uint8_t address, uint16_t lines);

uint32_t LS_CALL ls_getlinesperframe(uint8_t address, uint16_t* lines);
```

Legacy API - Linux shared library

```
int32_t ls_setlinesperframe(uint8_t address, uint16_t lines, uint32_t timeout_ms);

int32_t ls_getlinesperframe(uint8_t address, uint16_t* lines, uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended functions.
address	In	8-bit LISM-PI26xx I2C address.

Parameter	Direction	Description
lines	In/Out	Number of lines assigned to one frame.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy functions.

Return value. 0 on success; otherwise an I2C, timeout or handle error.

Quadrature Encoder Count - Set / Get

Writes or reads the 32-bit QUAD_COUNT register. When quadrature count control is enabled in MODE_CONFIG, the module generates an acquisition event after the configured number of encoder counts in the selected direction.

When quadrature count control is disabled, each valid encoder count can trigger one acquisition line and the QUAD_COUNT value is not used for division.

Extended API - Windows and Linux

```
ls_result_t LS_CALL ls_setquadcount_ex(
    ls_handle_t handle,
    uint8_t address,
    uint32_t count,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_getquadcount_ex(
    ls_handle_t handle,
    uint8_t address,
    uint32_t* count,
    uint32_t timeout_ms
);
```

Legacy API - Windows DLL

```
uint32_t LS_CALL ls_setquadcount(uint8_t address, uint32_t count);

uint32_t LS_CALL ls_getquadcount(uint8_t address, uint32_t* count);
```

Legacy API - Linux shared library

```
int32_t ls_setquadcount(uint8_t address, uint32_t count, uint32_t timeout_ms);

int32_t ls_getquadcount(uint8_t address, uint32_t* count, uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended functions.
address	In	8-bit LISM-PI26xx I2C address.
count	In/Out	Number of quadrature-encoder events per acquisition trigger.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy functions.

Return value. 0 on success; otherwise an I2C, timeout or handle error.

Internal Software Trigger Period - Set / Get

Writes or reads the 32-bit SOFT_TRIGGER_PERIOD register. The value is expressed in microseconds and defines the interval between internally generated trigger events in software-trigger mode.

The period must be longer than the complete sensor start-pulse period and must allow the selected sensor to finish integration and readout before the next trigger.

Extended API - Windows and Linux

```
ls_result_t LS_CALL ls_setsofttriggertime_ex(
    ls_handle_t handle,
    uint8_t address,
    uint32_t time_value,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_getsofttriggertime_ex(
    ls_handle_t handle,
    uint8_t address,
    uint32_t* time_value,
    uint32_t timeout_ms
);
```

```
Legacy API - Windows DLL
uint32_t LS_CALL ls_setsofttriggertime(uint8_t address, uint32_t time_value);

uint32_t LS_CALL ls_getsofttriggertime(uint8_t address, uint32_t* time_value);

Legacy API - Linux shared library
int32_t ls_setsofttriggertime(uint8_t address, uint32_t time_value, uint32_t timeout_ms);

int32_t ls_getsofttriggertime(uint8_t address, uint32_t* time_value, uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended functions.
address	In	8-bit LISM-PI26xx I2C address.
time_value	In/Out	Trigger period in microseconds.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy functions.

Return value. 0 on success; otherwise an I2C, timeout or handle error.

Trigger Output Delay - Set / Get

Writes or reads the 32-bit TRIGGER_OUTPUT_DELAY register. The value delays the trigger-output pulse relative to the start of the integration/acquisition cycle and is expressed in microseconds.

Use the delay to synchronize an external system with the sensor integration sequence.

```
Extended API - Windows and Linux
ls_result_t LS_CALL ls_settriggerdelay_ex(
    ls_handle_t handle,
    uint8_t address,
    uint32_t delay_value,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_gettriggerdelay_ex(
    ls_handle_t handle,
    uint8_t address,
    uint32_t* delay_value,
    uint32_t timeout_ms
);

Legacy API - Windows DLL
uint32_t LS_CALL ls_settriggerdelay(uint8_t address, uint32_t delay_value);

uint32_t LS_CALL ls_gettriggerdelay(uint8_t address, uint32_t* delay_value);

Legacy API - Linux shared library
int32_t ls_settriggerdelay(uint8_t address, uint32_t delay_value, uint32_t timeout_ms);

int32_t ls_gettriggerdelay(uint8_t address, uint32_t* delay_value, uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended functions.
address	In	8-bit LISM-PI26xx I2C address.
delay_value	In/Out	Trigger-output delay in microseconds.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy functions.

Return value. 0 on success; otherwise an I2C, timeout or handle error.

Trigger Output Width - Set / Get

Writes or reads the 32-bit TRIGGER_OUTPUT_WIDTH register. The value defines the active duration of the trigger-output pulse in microseconds.

The sum of trigger delay and pulse width should fit within the acquisition/start-pulse period. If it extends beyond the cycle, the output is forced inactive before the next integration cycle.

Extended API - Windows and Linux

```
ls_result_t LS_CALL ls_settriggerwidth_ex(
    ls_handle_t handle,
    uint8_t address,
    uint32_t width_value,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_gettriggerwidth_ex(
    ls_handle_t handle,
    uint8_t address,
    uint32_t* width_value,
    uint32_t timeout_ms
);
```

Legacy API - Windows DLL

```
uint32_t LS_CALL ls_settriggerwidth(uint8_t address, uint32_t width_value);

uint32_t LS_CALL ls_gettriggerwidth(uint8_t address, uint32_t* width_value);
```

Legacy API - Linux shared library

```
int32_t ls_settriggerwidth(uint8_t address, uint32_t width_value, uint32_t timeout_ms);

int32_t ls_gettriggerwidth(uint8_t address, uint32_t* width_value, uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended functions.
address	In	8-bit LISM-PI26xx I2C address.
width_value	In/Out	Trigger-output active width in microseconds.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy functions.

Return value. 0 on success; otherwise an I2C, timeout or handle error.

Pixel Count - Set / Get

Writes or reads the 16-bit PIXEL_COUNT register. The value defines how many 16-bit pixel values are output for each line and therefore controls the duration of LINE_VALID and the basic line byte count.

One line normally contains pixel_count x 2 bytes, and the host packet length must also include the CFG1 image-grouping factor. Because the USB interface board generates its packet size from PixelCount, changing this register requires the hardware-reset and reopen sequence described below.

Extended API - Windows and Linux

```
ls_result_t LS_CALL ls_setpixelcount_ex(
    ls_handle_t handle,
    uint8_t address,
    uint16_t pixel_count,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_getpixelcount_ex(
    ls_handle_t handle,
    uint8_t address,
    uint16_t* pixel_count,
    uint32_t timeout_ms
);
```

Legacy API - Windows DLL

```
uint32_t LS_CALL ls_setpixelcount(uint8_t address, uint16_t pixel_count);

uint32_t LS_CALL ls_getpixelcount(uint8_t address, uint16_t* pixel_count);
```

Legacy API - Linux shared library

```
int32_t ls_setpixelcount(uint8_t address, uint16_t pixel_count, uint32_t timeout_ms);

int32_t ls_getpixelcount(uint8_t address, uint16_t* pixel_count, uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended functions.
address	In	8-bit LISM-PI26xx I2C address.
pixel_count	In/Out	Number of 16-bit pixels transferred for each line.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy functions.

Return value. 0 on success; otherwise an I2C, timeout or handle error.

Important: After ls_setpixelcount or ls_setpixelcount_ex changes PixelCount, physically disconnect and reconnect the USB cable to reset the USB interface board. Closing and reopening only the software handle is not sufficient. After reconnection, re-enumerate and reopen the interface, then verify or update the host packet length before restarting acquisition.

Edges Before Data - Set / Get

Writes or reads the 8-bit EDGES_BEFORE_DATA register. The value defines the number of sensor/data-clock edges between the sensor timing reference and the first valid pixel transferred by the parallel interface.

This parameter aligns the valid pixel window with the installed sensor and should normally use the sensor-specific factory value.

Extended API - Windows and Linux

```
ls_result_t LS_CALL ls_setedgedelay_ex(
    ls_handle_t handle,
    uint8_t address,
    uint8_t edge_delay,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_getedgedelay_ex(
    ls_handle_t handle,
    uint8_t address,
    uint8_t* edge_delay,
    uint32_t timeout_ms
);
```

Legacy API - Windows DLL

```
uint32_t LS_CALL ls_setedgedelay(uint8_t address, uint8_t edge_delay);

uint32_t LS_CALL ls_getedgedelay(uint8_t address, uint8_t* edge_delay);
```

Legacy API - Linux shared library

```
int32_t ls_setedgedelay(uint8_t address, uint8_t edge_delay, uint32_t timeout_ms);

int32_t ls_getedgedelay(uint8_t address, uint8_t* edge_delay, uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended functions.
address	In	8-bit LISM-PI26xx I2C address.
edge_delay	In/Out	Number of clock edges before valid pixel data.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy functions.

Return value. 0 on success; otherwise an I2C, timeout or handle error.

ADC Full-Scale Selection - Set / Get

Writes or reads the ADC_FULL_SCALE register. Value 0 selects a 1.6 V full-scale range and value 1 selects a 2.0 V full-scale range.

The selected range affects the analog input span and should be chosen together with gain, offset and input bias so the sensor signal uses the ADC range without clipping.

Extended API - Windows and Linux

```
ls_result_t LS_CALL ls_setadcvref_ex(
    ls_handle_t handle,
    uint8_t address,
```

```
    uint8_t vref,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_getadcvref_ex(
    ls_handle_t handle,
    uint8_t address,
    uint8_t* vref,
    uint32_t timeout_ms
);
```

Legacy API - Windows DLL

```
uint32_t LS_CALL ls_setadcvref(uint8_t address, uint8_t vref);

uint32_t LS_CALL ls_getadcvref(uint8_t address, uint8_t* vref);
```

Legacy API - Linux shared library

```
int32_t ls_setadcvref(uint8_t address, uint8_t vref, uint32_t timeout_ms);

int32_t ls_getadcvref(uint8_t address, uint8_t* vref, uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended functions.
address	In	8-bit LISM-PI26xx I2C address.
vref	In/Out	0 = 1.6 V full scale; 1 = 2.0 V full scale.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy functions.

Return value. 0 on success; otherwise an I2C, timeout or handle error.

ADC Gain - Set / Get

Writes or reads the ADC_GAIN register. The gain parameter is a 6-bit unsigned code G in the range 0...63; bits [7:6] are reserved and must be zero. The internal ADC gain is Gain_ADC = 76 / (76 - G), producing approximately 1.00x at G = 0 and 5.85x at G = 63.

The complete signal-path gain also includes the sensor-board-specific analog front-end scaling: Gain_Total = Gain_ADC x ADC_GAIN_MULT / ADC_GAIN_DIV. Read the multiplier and divisor with ls_getgainmult and ls_getgaindiv.

Extended API - Windows and Linux

```
ls_result_t LS_CALL ls_setadcgain_ex(
    ls_handle_t handle,
    uint8_t address,
    uint8_t gain,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_getadcgain_ex(
    ls_handle_t handle,
    uint8_t address,
    uint8_t* gain,
    uint32_t timeout_ms
);
```

Legacy API - Windows DLL

```
uint32_t LS_CALL ls_setadcgain(uint8_t address, uint8_t gain);

uint32_t LS_CALL ls_getadcgain(uint8_t address, uint8_t* gain);
```

Legacy API - Linux shared library

```
int32_t ls_setadcgain(uint8_t address, uint8_t gain, uint32_t timeout_ms);

int32_t ls_getadcgain(uint8_t address, uint8_t* gain, uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended functions.
address	In	8-bit LISM-PI26xx I2C address.

Parameter	Direction	Description
gain	In/Out	6-bit gain code G in the range 0...63. Bits [7:6] must be zero.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy functions.

Return value. 0 on success; otherwise an I2C, timeout or handle error.

Bits	Allowed value	Meaning
5:0	0...63	Straight-binary ADC gain code G. The analog gain is calculated as 76 / (76 - G).
7:6	0	Reserved; must be zero.

ADC Gain Scaling - Get Multiplier / Divisor

Reads the 16-bit gain multiplier and divisor stored by the module. They allow software to convert the gain register code according to the analog front-end fitted to the hardware.

Treat a zero or unavailable divisor as invalid and do not perform a division until both values have been read successfully.

Extended API - Windows and Linux

```
ls_result_t LS_CALL ls_getgainmult_ex(
    ls_handle_t handle,
    uint8_t address,
    uint16_t* multiplier,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_getgaindiv_ex(
    ls_handle_t handle,
    uint8_t address,
    uint16_t* divisor,
    uint32_t timeout_ms
);
```

Legacy API - Windows DLL

```
uint32_t LS_CALL ls_getgainmult(uint8_t address, uint16_t* multiplier);

uint32_t LS_CALL ls_getgaindiv(uint8_t address, uint16_t* divisor);
```

Legacy API - Linux shared library

```
int32_t ls_getgainmult(uint8_t address, uint16_t* multiplier, uint32_t timeout_ms);

int32_t ls_getgaindiv(uint8_t address, uint16_t* divisor, uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended functions.
address	In	8-bit LISM-PI26xx I2C address.
multiplier	Out	Receives the gain scaling multiplier.
divisor	Out	Receives the gain scaling divisor.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy functions.

Return value. 0 on success; otherwise an I2C, timeout or handle error.

ADC Offset - Set / Get

Writes or reads the ADC_OFFSET register. The offset is a 9-bit sign-magnitude value stored in a 16-bit word. Bit 8 selects the sign, bits [7:0] contain the magnitude and bits [15:9] are reserved and must be zero.

Magnitude 0...255 covers approximately 0...250 mV, or about 0.98 mV per step. The offset is applied in the analog path before digitization and should be adjusted together with ADC gain, full-scale range and input bias.

Extended API - Windows and Linux

```
ls_result_t LS_CALL ls_setadcoffset_ex(
    ls_handle_t handle,
    uint8_t address,
    uint16_t offset,
    uint32_t timeout_ms
);
ls_result_t LS_CALL ls_getadcoffset_ex(
```

```
    ls_handle_t handle,
    uint8_t address,
    uint16_t* offset,
    uint32_t timeout_ms
);
Legacy API - Windows DLL
uint32_t LS_CALL ls_setadcoffset(uint8_t address, uint16_t offset);
uint32_t LS_CALL ls_getadcoffset(uint8_t address, uint16_t* offset);
Legacy API - Linux shared library
int32_t ls_setadcoffset(uint8_t address, uint16_t offset, uint32_t timeout_ms);
int32_t ls_getadcoffset(uint8_t address, uint16_t* offset, uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended functions.
address	In	8-bit LISM-PI26xx I2C address.
offset	In/Out	9-bit sign-magnitude ADC offset code stored in a 16-bit value. Reserved bits [15:9] must be zero.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy functions.

Return value. 0 on success; otherwise an I2C, timeout or handle error.

Bits	Values	Meaning
7:0	0...255	Offset magnitude. 255 corresponds to approximately 250 mV.
8	0 / 1	Sign: 0 = positive offset; 1 = negative offset.
15:9	0	Reserved; must be zero.

ADC Input Bias - Set / Get

Writes or reads the ADC_INPUT_BIAS register. The bias parameter is a 10-bit unsigned DAC code stored in a 16-bit word. Bits [9:0] contain values 0...1023 and bits [15:10] are reserved and must be zero.

The corresponding bias voltage is approximately V_bias = bias / 1023 x 2.5 V. This DAC-generated voltage is subtracted from the sensor signal before gain and digitization and provides coarse baseline alignment.

```
Extended API - Windows and Linux
ls_result_t LS_CALL ls_setadcbias_ex(
    ls_handle_t handle,
    uint8_t address,
    uint16_t bias,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_getadcbias_ex(
    ls_handle_t handle,
    uint8_t address,
    uint16_t* bias,
    uint32_t timeout_ms
);

Legacy API - Windows DLL
uint32_t LS_CALL ls_setadcbias(uint8_t address, uint16_t bias);

uint32_t LS_CALL ls_getadcbias(uint8_t address, uint16_t* bias);

Legacy API - Linux shared library
int32_t ls_setadcbias(uint8_t address, uint16_t bias, uint32_t timeout_ms);

int32_t ls_getadcbias(uint8_t address, uint16_t* bias, uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended functions.
address	In	8-bit LISM-PI26xx I2C address.
bias	In/Out	10-bit DAC code in the range 0...1023, stored in a 16-bit value. Bits [15:10] must be zero.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy functions.

Return value. 0 on success; otherwise an I2C, timeout or handle error.

Bits	Allowed value	Meaning
9:0	0...1023	Bias DAC code. 0 corresponds to approximately 0 V and 1023 to approximately 2.5 V.
15:10	0	Reserved; must be zero.

Clock and Timing Generator State - ls_getctgstate

Reads the RESUME_GENERATOR state register. A value of 0xAA indicates that Clock and Timing Generator operation is resumed. A non-0xAA value indicates the suspended/reset state.

This state is separate from the START_ACQUISITION value returned by ls_getstate.

Extended API - Windows and Linux
```
ls_result_t LS_CALL ls_getctgstate_ex(
    ls_handle_t handle,
    uint8_t address,
    uint8_t* state,
    uint32_t timeout_ms
);
```

Legacy API - Windows DLL
```
uint32_t LS_CALL ls_getctgstate(uint8_t address, uint8_t* state);
```

Legacy API - Linux shared library
```
int32_t ls_getctgstate(uint8_t address, uint8_t* state, uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended function.
address	In	8-bit LISM-PI26xx I2C address.
state	Out	Receives the RESUME_GENERATOR register value.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy functions.

Return value. 0 on success; otherwise an I2C, timeout or handle error.

5.5 Settings, Status and Identification

Save / Reload / Reset Settings

ls_savesettings stores the current module configuration, except the I2C slave address, in EEPROM. ls_reloadsettings reads the saved values and reinitializes the hardware configuration. ls_resetsettings restores the module default settings.

These functions send command-only register writes. The separate ls_saveslaveaddress function is required to store a new I2C address.

Extended API - Windows and Linux
```
ls_result_t LS_CALL ls_savesettings_ex(
    ls_handle_t handle,
    uint8_t address,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_reloadsettings_ex(
    ls_handle_t handle,
    uint8_t address,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_resetsettings_ex(
    ls_handle_t handle,
    uint8_t address,
    uint32_t timeout_ms
);
```

Legacy API - Windows DLL
```
uint32_t LS_CALL ls_savesettings(uint8_t address);

uint32_t LS_CALL ls_reloadsettings(uint8_t address);

uint32_t LS_CALL ls_resetsettings(uint8_t address);
```

Legacy API - Linux shared library
```
int32_t ls_savesettings(uint8_t address, uint32_t timeout_ms);

int32_t ls_reloadsettings(uint8_t address, uint32_t timeout_ms);

int32_t ls_resetsettings(uint8_t address, uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended functions.
address	In	8-bit LISM-PI26xx I2C address.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy functions.

Return value. 0 when the command was transmitted successfully; otherwise an I2C, timeout or handle error.

Important: EEPROM operations require internal processing time. The module can temporarily NACK a following I2C command; wait briefly and retry if required.

Initialization Status / Internal Communication Result

ls_getinitstatus reads INIT_STATUS (0xEE), an 8-bit readiness field produced by the LISM-PI26xx module after power-on, reload or reset. Bits 0...3 report detected internal components and bits 4...7 report successful initialization of the associated subsystems.

ls_getcomresult reads COM_RESULT (0xEF), an 8-bit status field for the most recent internal communication performed inside the LISM-PI26xx module. It reports the result of communication between the module MCU and the selected internal component or subsystem; it does not describe communication between the host computer and the USB interface. The function return value reports whether the current USB/I2C register read succeeded, while the returned byte describes the preceding internal module operation.

Extended API - Windows and Linux
```
ls_result_t LS_CALL ls_getinitstatus_ex(
    ls_handle_t handle,
    uint8_t address,
    uint8_t* status,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_getcomresult_ex(
    ls_handle_t handle,
    uint8_t address,
    uint8_t* result_value,
    uint32_t timeout_ms
);
```

Legacy API - Windows DLL
```
uint32_t LS_CALL ls_getinitstatus(uint8_t address, uint8_t* status);

uint32_t LS_CALL ls_getcomresult(uint8_t address, uint8_t* result_value);
```

Legacy API - Linux shared library
```
int32_t ls_getinitstatus(uint8_t address, uint8_t* status, uint32_t timeout_ms);

int32_t ls_getcomresult(uint8_t address, uint8_t* result_value, uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended functions.
address	In	8-bit LISM-PI26xx I2C address.
status / result_value	Out	Receives the raw bit-coded INIT_STATUS byte or the internal module COM_RESULT byte described below.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy functions.

Return value. 0 on successful register access; otherwise an I2C, timeout or handle error.

INIT_STATUS bit layout

Bit	Meaning when set to 1
0	Sensor-board EEPROM (HW1) detected.
1	DAC detected.
2	Processor-board EEPROM (HW2) detected.
3	Clock and Timing Generator detected.
4	Internal I2C bus initialized successfully.
5	DAC initialized successfully.
6	ADC initialized successfully.
7	Clock and Timing Generator initialized successfully.

COM_RESULT bit layout

Bits	Internal target	Meaning of returned bit
0	Clock and Timing Generator	1 = most recent internal access successful; 0 = error.
1	ADC	1 = most recent internal access successful; 0 = error.
2	DAC	1 = most recent internal access successful; 0 = error.
3	Processor-board EEPROM (HW2)	1 = most recent internal access successful; 0 = error.
4	Sensor-board EEPROM (HW1)	1 = most recent internal access successful; 0 = error.
7:5	Reserved	Reserved; ignore when read.

MCU and CTG Firmware Versions

Reads the 16-bit firmware version of the LISM-PI26xx communication MCU or Clock and Timing Generator. These are module firmware versions and are independent of the host-library version returned by ls_libversion.

The version format is defined by the module firmware. Applications commonly display the high and low bytes as major and minor components.

Extended API - Windows and Linux

```
ls_result_t LS_CALL ls_getmcuversion_ex(
    ls_handle_t handle,
    uint8_t address,
    uint16_t* version,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_getctgversion_ex(
    ls_handle_t handle,
    uint8_t address,
    uint16_t* version,
    uint32_t timeout_ms
);
```

Legacy API - Windows DLL

```
uint32_t LS_CALL ls_getmcuversion(uint8_t address, uint16_t* version);

uint32_t LS_CALL ls_getctgversion(uint8_t address, uint16_t* version);
```

Legacy API - Linux shared library

```
int32_t ls_getmcuversion(uint8_t address, uint16_t* version, uint32_t timeout_ms);

int32_t ls_getctgversion(uint8_t address, uint16_t* version, uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended functions.
address	In	8-bit LISM-PI26xx I2C address.
version	Out	Receives the 16-bit firmware version value.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy functions.

Return value. 0 on success; otherwise an I2C, timeout or handle error.

Hardware Identification and Hardware Versions

Reads the sensor-board identifier, sensor-board hardware version or processor/control-board hardware version from the LISM-PI26xx module.

These values can be used for logging, compatibility checks and hardware-specific configuration decisions. They describe the module connected through the selected USB interface, not the host USB interface hardware revision.

Extended API - Windows and Linux

```
ls_result_t LS_CALL ls_gethw1id_ex(
    ls_handle_t handle,
    uint8_t address,
    uint16_t* hardware_id,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_gethw1version_ex(
    ls_handle_t handle,
    uint8_t address,
    uint16_t* version,
    uint32_t timeout_ms
);

ls_result_t LS_CALL ls_gethw2version_ex(
    ls_handle_t handle,
    uint8_t address,
    uint16_t* version,
    uint32_t timeout_ms
);
```

Legacy API - Windows DLL

```
uint32_t LS_CALL ls_gethw1id(uint8_t address, uint16_t* hardware_id);

uint32_t LS_CALL ls_gethw1version(uint8_t address, uint16_t* version);

uint32_t LS_CALL ls_gethw2version(uint8_t address, uint16_t* version);
```

Legacy API - Linux shared library

```
int32_t ls_gethw1id(uint8_t address, uint16_t* hardware_id, uint32_t timeout_ms);

int32_t ls_gethw1version(uint8_t address, uint16_t* version, uint32_t timeout_ms);

int32_t ls_gethw2version(uint8_t address, uint16_t* version, uint32_t timeout_ms);
```

Parameters

Parameter	Direction	Description
handle	In	Open device handle for the extended functions.
address	In	8-bit LISM-PI26xx I2C address.
hardware_id	Out	Receives the sensor-board identifier.
version	Out	Receives the selected 16-bit hardware version.
timeout_ms	In	Maximum control-transfer time for the extended and Linux legacy functions.

Return value. 0 on success; otherwise an I2C, timeout or handle error.

6 Error Codes

Legacy and *_ex status-returning functions use the same logical public error symbols. The library defines the public codes listed below, including endpoint-timeout and acquisition-thread-priority validation errors. Numeric values can differ between Windows and Linux, so applications should compare against the platform header and use ls_geterrorstring for diagnostic output.

A status-returning function may also pass through a native backend error when no exact public-library equivalent exists. ls_enumdevices is count-valued and therefore uses a special Windows failure sentinel of -1 instead of returning a small positive native error.

Symbol	Windows	Linux	Meaning
LS_OK	0	0	Success
LS_ERR_NO_DEVICES_ATTACHED	0x20000001	-10001	No matching devices found
LS_ERR_DEVICE_INDEX_OUT_OF_RANGE	0x20000002	-10002	Invalid enumeration index
LS_ERR_SERIAL_NUM_NOT_FOUND	0x20000003	-10003	Serial number not found
LS_ERR_INVALID_HANDLE	0x00000006	-10004	Invalid, closed or unknown device handle
LS_ERR_DEVICE_ACTIVE	0x20000011	-10005	Device is open or busy
LS_ERR_CONTROL_TRANSFER	0x20000015	-10006	USB transfer failed or was incomplete
LS_ERR_TIMEOUT	0x000005B4	-10007	Operation timed out
LS_ERR_DEVICE_ALREADY_OPEN	0x20000013	-10008	Physical device is already open
LS_ERR_NOT_IMPLEMENTED	0x20000014	-10009	Operation is not implemented by the selected backend
LS_ERR_ALLOCATE_THREADPARAM_MEM	0x20000004	-10010	Device, thread or FIFO allocation failed
LS_ERR_PATH_INVALID_HANDLE	0x20000006	-10012	Invalid device path or setup handle
LS_ERR_INVALID_DATALEN	0x21000001	-10051	Invalid buffer/transfer length or host FIFO smaller than packet length
LS_ERR_I2C_NACK	0x21000004	-10052	I2C target returned NACK
LS_ERR_I2C_ERROR	0x21000005	-10053	I2C transfer error
LS_ERR_I2C_UNKNOWN	0x21000006	-10054	Unknown I2C status
LS_ERR_I2C_MUX_DISABLED	0x21000007	-10055	I2C multiplexer disabled or invalid
LS_ERR_INVALID_PACKETLENGTH	0x21000002	-10056	Invalid packet length
LS_ERR_CMD_NOT_SUPPORTED	0x21000003	-10057	Command or parameter is not supported
LS_ERR_PIPE_CANCELLED	0x0000006D	-10058	Host FIFO operation cancelled because the device is closing
LS_ERR_INVALID_TIMEOUT	0x21000008	-10059	Endpoint timeout must be greater than zero
LS_ERR_INVALID_THREAD_PRIORITY	0x21000009	-10060	Invalid portable acquisition-thread priority
LS_ERR_THREAD_PRIORITY_FAILED	0x2100000A	-10061	Acquisition-thread priority could not be applied

Native Backend Errors

Windows normalizes ERROR_INVALID_HANDLE to LS_ERR_INVALID_HANDLE, timeout errors to LS_ERR_TIMEOUT and intentional host-FIFO cancellation during close to LS_ERR_PIPE_CANCELLED. Other Win32, WinUSB, SetupAPI or driver errors are returned unchanged so their diagnostic meaning is preserved.

Linux uses two negative ranges: -1 through -9999 for native libusb errors and -10000 or below for USB LISM-PI26xx library errors. LIBUSB_ERROR_TIMEOUT is normalized to LS_ERR_TIMEOUT; other libusb errors remain unchanged. Positive POSIX errno values are not returned by the public Linux API.

Close and FIFO Error Semantics

Condition	Result
No open device, unknown handle or already closed handle	LS_ERR_INVALID_HANDLE
Blocked pipe wait/read interrupted by an intentional CloseDevice call	LS_ERR_PIPE_CANCELLED
CloseDevice cannot complete within its internal wait period	LS_ERR_TIMEOUT
Second close overlaps an already active close attempt	LS_ERR_DEVICE_ACTIVE
Pipe call starts after the device has entered the closing state	LS_ERR_PIPE_CANCELLED
Physical USB disconnect	Original native backend disconnect error, for example LIBUSB_ERROR_NO_DEVICE on Linux

Error-Handling Recommendations

- Check every result-returning open, close, I2C, CTG and pipe function.
- Log both the numeric result and ls_geterrorstring(result).
- Treat LS_ERR_PIPE_CANCELLED as the expected result of a blocked FIFO operation interrupted by an intentional device close.
- Treat LS_ERR_INVALID_TIMEOUT and LS_ERR_INVALID_THREAD_PRIORITY as application configuration errors; correct the supplied value before retrying.
- After a physical USB disconnect, discard the old context, re-enumerate and open a new context; do not reuse the old handle.
- Do not assume that Windows and Linux use the same numeric value, or that every native backend error appears in the public-code table.

7 Revision History

Revision	Date	Description
1.0	July 2026	Initial release of the combined Windows/Linux library manual covering the legacy and *_ex APIs.