

LHandProLib SDK Manual

LHandProLib SDK Manual

Version: 1.5

Author: Leadtron

Target Platforms: Windows/Linux, supporting EtherCAT/CANFD/RS485 communication protocols

Introduction

`LHandProLib` is a C++ SDK for controlling dexterous hands, supporting multiple communication protocols (EtherCAT, CANFD, RS485). It offers core functionalities including dexterous hand degree-of-freedom (DoF) control, sensor data acquisition, homing, and status monitoring. This manual provides detailed descriptions of all interfaces, enumeration types, and usage examples included in the SDK.

Namespace

All functionalities are defined within the `lhplib` namespace

代码块

```
1 namespace lhplib { ... }
```

Error Codes (LER)

Error Code	Macro Definition	Description
0	<code>LER_NONE</code>	Execution Successful
1	<code>LER_PARAMETER</code>	Invalid Parameter
2	<code>LER_KEY_FUNC_UNINIT</code>	Key Function Not Initialized (e.g., callback not set)

3	LER_GET_CONFIGURATION	Failed to Read Configuration
4	LER_COMM_CONNECT	Communication Connection Error
5	LER_COMM_SEND	Communication Transmission Error
6	LER_COMM_RECV	Communication Reception Error
7	LER_COMM_DATA_FORMAT	Communication Data Format Error
8	LER_INVALID_PATH	Invalid File Path
9	LER_LOG_SAVE_FAIL	Failed to Save Log File
10	LER_DATA_ANOMALY	Data Anomaly Error
11	LER_NOT_HOME	Homing Not Completed Error
999	LER_UNKNOWN	Unknown Error

Degree of Freedom (DoF) Enumeration

Enumeration Value Description	Description
LAC_DOF_6	6 DoF (Default)
LAC_DOF_15	15 DoF

Communication Type (LCN) Enumeration

Enumeration Value Description	Description
LCN_ECAT	EtherCAT Communication
LCN_CANFD	CANFD Communication
LCN_RS485	RS485 Communication

Control Mode (LCM) Enumeration

Enumeration Value Description	Description
LCM_POSITION	Position Control
LCM_VELOCITY	Velocity Control
LCM_TORQUE	Torque Control
LCM_VEL_TOR	Velocity + Torque Hybrid Control
LCM_POS_TOR	Position + Torque Hybrid Control
LCM_HOME	Homing Mode

Torque in place control mode (LTC) Enumeration

Enumeration Value Description	Description
LTC_REACHED_HOLD	Maintain the torque after it is in place
LTC_REACHED_STOP	Stop when the torque is in place

Operating State (LST) Enumeration

Enumeration Value Description	Description
LST_STOPPED	Normal Stop State
LST_RUNNING	Normal Operating State
LST_ALARM	Alarm Stop State
LST_POS_LIMIT	Positive Limit State
LST_NEG_LIMIT	Negative Limit State

LST_BOTH_LIMIT	Both Positive and Negative Limits Triggered
LST_EMG_STOP	Emergency Stop State
LST_HOMING	Homing Operating State

Sensor ID (LSS) Enumeration

Enumeration Value Description	Description
LSS_FINGER_1_1	Thumb Tip
LSS_FINGER_1_2	Thumb Pad
LSS_FINGER_2_1	Index Finger Tip
LSS_FINGER_2_2	Index Finger Pad
LSS_FINGER_3_1	Middle Finger Tip
LSS_FINGER_3_2	Middle Finger Pad
LSS_FINGER_4_1	Ring Finger Tip
LSS_FINGER_4_2	Ring Finger Pad
LSS_FINGER_5_1	Little Finger Tip
LSS_FINGER_5_2	Little Finger Pad
LSS_HAND_PALM	Palm Sensor
LSS_MAX_COUNT	Maximum Number of Sensors (for array allocation)

Hand Orientation (LDR) Enumeration

Enumeration Value Description	Description
LDR_HAND_RIGHT	Right Hand (Default)
LDR_HAND_LEFT	Left Hand

Function Pointer Types

Enumeration Value Description	Description
LogAddCallback	Log Callback Function: void(*) (const char*)
ECSendDataCallback	EtherCAT RPDO Transmission Callback: bool(*) (const unsigned char*, unsigned int)
CANFDSendDataCallback	CANFD Transmission Callback: bool(*) (const unsigned char*, unsigned int)

LHandProLib Class Interface Description

Constructors and Destructor

LHandProLib()

Default Constructor: Initializes internal resources and private objects (per Pimpl idiom).

~LHandProLib()

Destructor: Releases resources and automatically invokes close().

Initialization and Shutdown

int initial(int mode)

Initializes the dexterous hand driver.

- **Parameter:**
 - mode : Communication type. Valid values LCN_ECATT / LCN_CANFD / LCN_RS485
- **Return Value:**
 - 0 : Execution successful
 - Others: Refer to LER_* error codes

- **Preconditions:**

- i. The underlying communication device is connected.
 - ii. Transmission callback is configured (`set_send_rpdo_callback`).
 - iii. Received data processing is configured (`set_tpdo_data_decode`).
-

```
void close()
```

Shuts down the driver, releases resources, and terminates all connections.

Communication Callback Configuration

```
void set_send_rpdo_callback(ECSendDataCallback callback)
```

Configures EtherCAT RPDO transmission callback (for output data).

- **Parameter:**

- `callback` : Function pointer with prototype `bool(const unsigned char* data, int size)`
 - `data` : RPDO data to be transmitted
 - `size` : Total data length
 - Return Value: `true` Successful, `false` Failed

- **Note:** Triggered by `get_pre_send_rpdo_data` (if configured).
-

```
void set_send_rpdo_callback_ex(void* callback_impl)
```

Configures EtherCAT transmission callback wrapped in `std::function`.

- **Parameter:**

- `callback_impl` : Pointer to `std::function<bool(const unsigned char*, unsigned int)>`

- **Note:** Lifecycle management is the user's responsibility.
-

```
void set_send_canfd_callback(CANFDSendDataCallback callback)
```

Configures CANFD transmission callback (for output data).

- **Parameter:**

- `callback` : Function pointer with prototype `bool(const unsigned char*, unsigned int)`
 - `data` : CANFD data to be transmitted
 - `size` : Total data length
 - Return Value: `true` Successful, `false` Failed
 - **Note:** Triggered by `get_pre_send_rpdo_data` (if configured).
-

```
void set_send_canfd_callback_ex(void* callback_impl)
```

Configures CANFD transmission callback wrapped in `std::function`.

- **Parameter:**
 - `callback_impl` : Pointer to `std::function<bool(const unsigned char*, unsigned int)>`
 - **Note:** Lifecycle management is the user's responsibility.
-

```
int get_pre_send_rpdo_data(unsigned char* data_ptr, int* io_size)
```

Retrieves RPDO data to be transmitted (for EtherCAT output area).

- **Parameter:**
 - `data_ptr` : Output buffer pointer
 - `nullptr` to only get required buffer size
 - `io_size` : Input - Buffer capacity; Output - Actually required or written byte count
 - **Return Value:**
 - `0` : Execution successful
 - Others: Refer to `LER_*` error codes
 - **Note:** Automatically triggers transmission if `set_send_rpdo_callback` is configured.
-

```
int set_tpdo_data_decode(const unsigned char* data_ptr, int data_size)
```

Decodes TPDO data received from EtherCAT (input data).

- **Parameter:**
 - `data_ptr` : TPDO data pointer

- `data_size` : Total data length (bytes)
 - **Return Value:**
 - `0` : Execution successful
 - Others: Refer to `LER_*` error codes
 - **Note:** Latest status can be retrieved via interfaces like `get_now_angle` after decoding.
-

DOF and Orientation Configuration

```
int set_dof_type(int dof_type)
```

Configures dexterous hand DOF type.

- **Parameter:**
 - `dof_type` : DOF type. Valid values: `LAC_DOF_6` `LAC_DOF_15`
 - **Return Value:**
 - `0` : Execution successful
 - Others: Refer to `LER_*` error codes
 - **Default Value:** `LAC_DOF_6`
-

```
int get_dof(int* total, int* active)
```

Retrieves current dexterous hand DOF information.

- **Parameter:**
 - `total` : Output parameter, returns total joint count (including passive joints)
 - `active` : Output parameter, returns active motor count
 - **Return Value:**
 - `0` : Execution successful
 - Others: Refer to `LER_*` error codes
-

```
int set_hand_direction(int dir)
```

Configures hand orientation (affects motion mapping logic).

- **Parameter:**
 - `dir` : Orientation. Valid values: `LDR_HAND_RIGHT` `LDR_HAND_LEFT`

- **Return Value:**

- 0 : Execution successful
 - Others: Refer to LER_* error codes
-

```
int get_hand_direction(int* dir)
```

Retrieves current hand orientation.

- **Parameter:**

- dir : Output parameter, receives orientation value: LDR_HAND_RIGHT
LDR_HAND_LEFT

- **Return Value:**

- 0 : Execution successful
 - Others: Refer to LER_* error codes
-

Motor Control Interfaces

```
int set_control_mode(int id, int mode = LCM_POSITION)
```

Configures motor control mode.

- **Parameter:**

- id : Motor ID. Valid range: [0, DOF]
 - id = 0 : Broadcast mode (configure all motors)
 - id ≥ 1 : Configure specified motor
- mode : Control mode. Refer to LCM_*

- **Return Value:**

- 0 : Execution successful
 - Others: Refer to LER_* error codes
-

```
int get_control_mode(int id, int* mode)
```

Gets the control mode of the specified motor.

- **Parameters:**

- id : Motor ID. Value range: [1, DOF]

- mode: Output parameter. Receives the current control mode

- **Return Value:**

- 0 : Execution successful
 - Others: Refer to LER_* error codes
-

```
int set_torque_control_mode(int id, int mode)
```

Sets the torque reaching control mode for the specified motor

- **Parameters:**

- id : Motor ID. Value range: [0, DOF]
 - id = 0 : Broadcast mode (configures all motors)
 - id ≥ 1 : Configures the specified motor
- mode : Torque control mode. Value references the LTC_* enumeration

- **Return Value:**

- 0 : Execution successful
 - Others: Refer to LER_* error codes
-

```
int get_torque_control_mode(int id, int* mode)
```

Gets the torque reaching control mode of the specified motor

- **Parameters:**

- id : Motor ID. Value range: [1, DOF]
- mode : Output parameter. Receives the current torque reaching control mode.

- **Return Value:**

- 0 : Execution successful
 - Others: Refer to LER_* error codes
-

```
int set_enable(int id, int enable)
```

Enables or disables the motor driver

- **Parameters:**

- id : Motor ID. Value range: [0, DOF]
 - id = 0 : Broadcast mode (applies to all motors)

- `id ≥ 1` : Applies to the specified motor
 - `enable` : Enable flag, `1` Enable, `0` Disable
 - **Return Value:**
 - `0` : Execution successful
 - Others: Refer to `LER_*` error codes
-

```
int get_enable(int id, int* enable)
```

Gets the current enable status of the motor

- **Parameters:**
 - `id` : Motor ID. Value range: `[1, DOF]`
 - `enable` : Output parameter, `1` Enabled, `0` Disabled
 - **Return Value:**
 - `0` : Execution successful
 - Others: Refer to `LER_*` error codes
-

```
int get_position_reached(int id, int* reached)
```

Gets whether the motor has reached the target position

- **Parameters:**
 - `id` : Motor ID. Value range: `[1, DOF]`
 - `reached` : Output parameter, `1` Position reached, `0` In motion
 - **Return Value:**
 - `0` : Execution successful
 - Others: Refer to `LER_*` error codes
-

```
int get_torque_reached(int id, int* reached)
```

Gets whether the motor has reached the target torque

- **Parameters:**
 - `id` : Motor ID. Value range: `[1, DOF]`
 - `reached` : Output parameter, `1` Torque reached, `0` Torque not reached
- **Return Value:**

- 0 : Execution successful
 - Others: Refer to LER_* error codes
-

```
int set_clear_alarm(int id = 0)
```

Clear motor alarm status

- **Parameters:**

- id : Motor ID. Value range: [0, DOF]
 - id = 0 : Broadcast to clear all motor alarms
 - id ≥ 1 : Clear alarm for the specified motor

- **Return Value:**

- 0 : Execution successful
 - Others: Refer to LER_* error codes
-

```
int get_now_alarm(int id, int* alarm)
```

Get the motor's current alarm status code

- **Parameters:**

- id : Motor ID. Value range: [1, DOF]
- alarm : Output parameter, receive alarm codes (refer to the hardware manual for specific interpretations)

- **Return Value:**

- 0 : Execution successful
 - Others: Refer to LER_* error codes
-

```
int home_motors(int id)
```

Initiate motor homing

- **Parameters:**

- id : Motor ID. Value range: [0, DOF]
 - id = 0 : Initiate homing for all motors
 - id ≥ 1 : Initiate homing for the specified motor

- **Return Value:**

- 0 : Execution successful
 - Others: Refer to LER_* error codes
-

```
int get_limit_target_angle(int id, float* max_angle, float* min_angle)
```

Get the upper and lower limits of the motor's target angle

- **Parameters:**

- id : Motor ID. Value range: [1, DOF]
- max_angle : Output parameter, Maximum allowable angle (Unit: °)
- min_angle : Output parameter, Minimum allowable angle (Unit: °)

- **Return Value:**

- 0 : Execution successful
 - Others: Refer to LER_* error codes
-

```
int set_target_angle(int id, float angle)
```

Set the motor's target angle

- **Parameters:**

- id : Motor ID. Value range: [0, DOF]
 - id = 0 : Broadcast mode (applies to all motors)
 - id ≥ 1 : Applies to the specified motor
- angle : target angle unit: ° (degrees)

- **Return Value:**

- 0 : Execution successful
- Others: Refer to LER_* error codes

- **Note:** The actual achievable range is determined by the mechanical structure.
-

```
int get_target_angle(int id, float* angle)
```

Get the last set target angle

- **Parameters:**

- id : Motor ID. Value range: [1, DOF]

- `angle` : Output parameter, Receives the target angle value (Unit: °)
 - **Return Value:**
 - `0` : Execution successful
 - Others: Refer to `LER_*` error codes
-

```
int set_target_position(int id, int position)
```

Set the motor's target position (stroke equivalent)

- **Parameters:**
 - `id` : Motor ID. Value range: `[0, DOF]`
 - `id = 0` : Broadcast mode (applies to all motors)
 - `id ≥ 1` : Applies to the specified motor
 - `position` : target position unit: equivalent, Value range `[0, 10000]`
 - `0` Indicates the start position
 - `10000` Indicates the full stroke
 - **Return Value:**
 - `0` : Execution successful
 - Others: Refer to `LER_*` error codes
 - **Note:** The actual stroke is determined by the mechanical structure and calibration
-

```
int get_target_position(int id, int* position)
```

Get the last set target position (stroke equivalent)

- **Parameters:**
 - `id` : Motor ID. Value range: `[1, DOF]`
 - `position` : Output parameter, Receives the target position value (Unit: equivalent; Value range: `[0, 10000]`)
 - **Return Value:**
 - `0` : Execution successful
 - Others: Refer to `LER_*` error codes
-

```
int set_velocity(int id, float velocity)
```

Set the motor's target speed (default unit: angular velocity (°/s))

- **Parameters:**

- `id` : Motor ID. Value range: `[0, DOF]`
 - `id = 0` : Broadcast mode (applies to all motors)
 - `id ≥ 1` : Applies to the specified motor
- `velocity` : Target speed unit: °/s

- **Return Value:**

- `0` : Execution successful
- Others: Refer to `LER_*` error codes

- **Note:** The actual speed is subject to hardware limitations; it is recommended to use `set_angular_velocity` for clear semantics
-

```
int get_velocity(int id, float* velocity)
```

Get the last set target speed (returns angular velocity by default)

- **Parameters:**

- `id` : Motor ID. Value range: `[1, DOF]`
- `velocity` : Output parameter, Receives the target speed value (Unit: °/s)

- **Return Value:**

- `0` : Execution successful
- Others: Refer to `LER_*` error codes

- **Note:** It is recommended to use `get_angular_velocity` for clear semantics
-

```
int set_angular_velocity(int id, float velocity)
```

Set the motor's target angular velocity (Unit: °/s)

- **Parameters:**

- `id` : Motor ID. Value range: `[0, DOF]`
 - `id = 0` : Broadcast mode (applies to all motors)
 - `id ≥ 1` : Applies to the specified motor
- `velocity` : Target angular velocity unit: °/s

- **Return Value:**

- 0 : Execution successful
 - Others: Refer to LER_* error codes
 - **Note:** This interface is consistent in functionality with set_velocity but more clearly named; it is recommended for priority use
-

```
int get_angular_velocity(int id, float* velocity)
```

Get the last set target speed(Unit: °/s)

- **Parameters:**
 - id : Motor ID. Value range: [1, DOF]
 - velocity : Output parameter, Receives the target angular velocity value (Unit: °/s)
 - **Return Value:**
 - 0 : Execution successful
 - Others: Refer to LER_* error codes
-

```
int set_position_velocity(int id, int velocity)
```

Set the motor's target speed (stroke equivalent speed)

- **Parameters:**
 - id : Motor ID. Value range: [0, DOF]
 - id = 0 : Broadcast mode (applies to all motors)
 - id ≥ 1 : Applies to the specified motor
 - velocity : Target speed unit: equivalent/s (i.e., equivalents per second)
 - **Return Value:**
 - 0 : Execution successful
 - Others: Refer to LER_* error codes
-

```
int get_position_velocity(int id, int* velocity)
```

Get the last set target speed(i.e., equivalents per second)

- **Parameters:**
 - id : Motor ID. Value range: [1, DOF]
 - velocity : Output parameter, Receives the target speed value (Unit: equivalent/s)

- **Return Value:**

- 0 : Execution successful
 - Others: Refer to LER_* error codes
-

```
int set_max_current(int id, int current)
```

Set the motor's maximum allowable current

- **Parameters:**

- id : Motor ID. Value range: [0, DOF]
 - id = 0 : Broadcast mode (applies to all motors)
 - id ≥ 1 : Applies to the specified motor
- current : Maximum current limit (Unit: %)

- **Return Value:**

- 0 : Execution successful
- Others: Refer to LER_* error codes

- **Note:** Used for overcurrent protection and as the upper limit for torque control
-

```
int get_max_current(int id, int* current)
```

Get the last set maximum current value

- **Parameters:**

- id : Motor ID. Value range: [1, DOF]
- current : Output parameter, Receives the maximum current value (Unit: %)

- **Return Value:**

- 0 : Execution successful
 - Others: Refer to LER_* error codes
-

```
int move_motors(int id = 0)
```

Drive the motor to execute the set motion command

- **Parameters:**

- id : Motor ID. Value range: [0, DOF]
 - id = 0 : Start motion for all motors

- `id ≥ 1` : Start motion for the specified motor
 - **Return Value:**
 - `0` : Execution successful
 - Others: Refer to `LER_*` error codes
 - **Note:** The target value must be set first (e.g., via `set_target_angle`)
-

Status and Feedback Interfaces

```
int get_now_status(int id, int* status)
```

Get the motor's current operation status

- **Parameters:**
 - `id` : Motor ID. Value range: `[1, DOF]`
 - `status` : Output parameter, Receives the operation status code (refer to the `LST_*` enumeration)
 - **Return Value:**
 - `0` : Execution successful
 - Others: Refer to `LER_*` error codes
-

```
int get_now_angle(int id, float* angle)
```

Get the motor's current angular position

- **Parameters:**
 - `id` : Motor ID. Value range: `[1, DOF]`
 - `angle` : Output parameter, Receives the current angle (Unit: °)
 - **Return Value:**
 - `0` : Execution successful
 - Others: Refer to `LER_*` error codes
-

```
int get_now_position(int id, int* position)
```

Get the motor's current stroke position (equivalent)

- **Parameters:**

- `id` : Motor ID. Value range: `[1, DOF]`
 - `position` : Output parameter, Receives the current stroke position (Unit: equivalent; Value range: `[0, 10000]`)
 - **Return Value:**
 - `0` : Execution successful
 - Others: Refer to `LER_*` error codes
-

```
int get_now_velocity(int id, float* velocity)
```

Get the motor's current actual speed

- **Parameters:**
 - `id` : Motor ID. Value range: `[1, DOF]`
 - `velocity` : Output parameter, Receives the current speed (Unit: °/s)
 - **Return Value:**
 - `0` : Execution successful
 - Others: Refer to `LER_*` error codes
 - **Note:** This interface returns angular velocity by default; it is recommended to use `get_now_angular_velocity` for clear semantics
-

```
int get_now_angular_velocity(int id, float* velocity)
```

Get the motor's current actual angular velocity (Unit: °/s)

- **Parameters:**
 - `id` : Motor ID. Value range: `[1, DOF]`
 - `velocity` : Output parameter, Receives the current angular velocity (Unit: °/s)
 - **Return Value:**
 - `0` : Execution successful
 - Others: Refer to `LER_*` error codes
-

```
int get_now_position_velocity(int id, int* velocity)
```

Get the motor's current actual speed (stroke equivalent speed)

- **Parameters:**

- `id` : Motor ID. Value range: `[1, DOF]`
 - `velocity` : Output parameter, Receives the current speed (Unit: equivalent/s)
 - **Return Value:**
 - `0` : Execution successful
 - Others: Refer to `LER_*` error codes
-

```
int get_now_current(int id, int* current)
```

Get the motor's actual current

- **Parameters:**
 - `id` : Motor ID. Value range: `[1, DOF]`
 - `current` : Output parameter, Receives the actual current value (Unit: %)
 - **Return Value:**
 - `0` : Execution successful
 - Others: Refer to `LER_*` error codes
-

Haptic Sensor Interfaces

```
int get_finger_sensor_pos(int sensor_id, float* x_values,  
float* y_values, int* io_count)
```

Get the haptic sensor's distribution coordinates (normalized `[0.0, 1.0]`) 。

- **Parameters:**
 - `sensor_id` : Sensor ID, Value references the `LSS_*` enumeration
 - `x_values` : Output parameter, X-coordinate array pointer (can be `nullptr` to retrieve only the count)
 - `y_values` : Output parameter, Y-coordinate array pointer
 - `io_count` : Input: Indicates buffer size; Output: Returns the actual number of elements
 - **Return Value:**
 - `0` : Execution successful
 - Others: Refer to `LER_*` error codes
-

```
int get_finger_pressure(int sensor_id, float* pressure_list,
int* io_count)
```

Get the sensor pressure value (Value range: [0.0, 1.0]) 。

- **Parameters:**

- `sensor_id` : Sensor ID
- `pressure_list` : Output parameter, Pressure value array pointer (can be `nullptr` to retrieve only the count)
- `io_count` : Input: Buffer size; Output: Actual number of entries

- **Return Value:**

- `0` : Execution successful
 - Others: Refer to `LER_*` error codes
-

```
int set_finger_pressure_reset(int sensor_id)
```

Reset the sensor reference value (with the difference calculated from this moment)

- **Parameters:**

- `sensor_id` : Sensor ID
 - `sensor_id = 0` : Reset all sensors
 - `sensor_id > 0` : Reset only the specified sensor

- **Return Value:**

- `0` : Execution successful
 - Others: Refer to `LER_*` error codes
-

```
int get_finger_normal_force(int sensor_id, float* normal_force)
```

Get the normal force (perpendicular to the surface)

- **Parameters:**

- `sensor_id` : Sensor ID
- `normal_force` : Output parameter, Receives the normal force

- **Return Value:**

- `0` : Execution successful
- Others: Refer to `LER_*` error codes

```
int get_finger_normal_force_ex(int sensor_id, float* normal_force)
```

Extended version of `get_finger_normal_force` with enhanced functionality

- **Parameters:**

- `sensor_id` : Sensor ID
- `normal_force` : Output parameter, Receives the normal force

- **Return Value:**

- `0` : Execution successful
 - Others: Refer to `LER_*` error codes
-

```
int get_finger_tangential_force(int sensor_id, float* tangential_force)
```

Get the tangential force (parallel to the surface)

- **Parameters:**

- `sensor_id` : Sensor ID
- `tangential_force` : Output parameter, Receives the tangential force

- **Return Value:**

- `0` : Execution successful
 - Others: Refer to `LER_*` error codes
-

```
int get_finger_tangential_force_ex(int sensor_id, float* tangential_force)
```

Extended version of `get_finger_tangential_force` with enhanced functionality

- **Parameters:**

- `sensor_id` : Sensor ID
- `tangential_force` : Output parameter, Receives the tangential force

- **Return Value:**

- `0` : Execution successful
- Others: Refer to `LER_*` error codes

```
int get_finger_force_direction(int sensor_id, float* force_direction)
```

Get the force direction

- **Parameters:**

- `sensor_id` : Sensor ID
- `force_direction` : Output parameter, Receives the direction angle (Value range [0, 360], Unit: °)

- **Return Value:**

- 0 : Execution successful
 - Others: Refer to `LER_*` error codes
-

```
int get_finger_force_direction_ex(int sensor_id, float* force_direction)
```

Extended version of `get_finger_force_direction` with enhanced functionality

- **Parameters:**

- `sensor_id` : Sensor ID
- `force_direction` : Output parameter, Receives the direction angle (Value range [0, 360], Unit: °)

- **Return Value:**

- 0 : Execution successful
 - Others: Refer to `LER_*` error codes
-

```
int get_finger_proximity(int sensor_id, float* proximity)
```

Get the proximity level (e.g., for proximity sensors)

- **Parameters:**

- `sensor_id` : Sensor ID
- `proximity` : Output parameter, Receives the proximity level (Value range: [0.0, 1.0])

- **Return Value:**

- 0 : Execution successful

- Others: Refer to `LER_*` error codes
-

```
int get_finger_proximity_ex(int sensor_id, float* proximity)
```

Extended version of `get_finger_proximity` with enhanced functionality

- **Parameters:**

- `sensor_id` : Sensor ID
- `proximity` : Output parameter, Receives the proximity level (Value range: `[0.0, 1.0]`)

- **Return Value:**

- `0` : Execution successful
 - Others: Refer to `LER_*` error codes
-

Log System

```
void log_on(bool on = false, int maxsize = 10000)
```

Enable or disable log printing functionality

- **Parameters:**

- `on` : `true` Enable logging, `false` Disable logging (default)
 - `maxsize` : Maximum cached log entries: Prevents memory overflow
-

```
int log_send(int* cmd, int count)
```

Set the filter list for transmit command addresses to be printed

- **Parameters:**

- `cmd` : Transmit command address array pointer
 - `nullptr` Print all transmitted data
- `count` : Number of array elements

- **Return Value:**

- `0` : Execution successful
 - Others: Refer to `LER_*` error codes
-

```
int log_recv(int* cmd, int count)
```

Set the filter list of receiving instruction addresses that need to be printed

- **Parameters:**

- `cmd` : ransmit command address array pointe
 - `nullptr` Print all transmitted data
- `count` : Number of array elements

- **Return Value:**

- `0` : Execution successful
 - Others: Refer to `LER_*` error codes
-

```
void log_reset(bool send = true, bool recv = true)
```

Set the filter list for receive command addresses to be printed

- **Parameters:**

- `send` : Whether to clear the transmit filter list
 - `recv` : Whether to clear the receive filter list
-

```
int log_save(const char* file_name)
```

Save the current logs to a file

- **Parameters:**

- `file_name` : File path

- **Return Value:**

- `0` : Execution successful
 - Others: Refer to `LER_*` error codes
-

```
void log_clear()
```

Clear the log records in the current memory.

```
void set_log_callback(LogAddCallback callback)
```

Set a log callback function (to receive log strings in real-time)

- **Parameters:**
- `callback`: Function pointer with the prototype `void(const char*)`
 - The input parameter is a log string (ending with `\n`)
- **Example:**

代码块

```
1  lhp_lib->set_log_callback([](const char* msg) {
2      printf("[SDK_LOG] %s", msg);
3  });
```

Example of Usage Process (EtherCAT Communication)

代码块

```
1  #include "LHandProLib.hpp"
2  #include <iostream>
3  #include <chrono>
4  #include <thread>
5  #include <memory>
6  #include <functional>
7  #include "EthercatMaster.h" // EtherCAT Main Station Library
8
9  int main() {
10     /***** Initialization *****/
11     // Initialize LHandProLib object
12     auto lhp_lib_ = std::make_shared<LHandProLib>();
13     // Initialize EtherCAT master object
14     auto ec_master_ = std::make_shared<EthercatMaster>();
15     // Initialize EtherCAT master
16     std::vector<std::string> names = ec_master_->scanNetworkInterfaces();
17     int channel_index = 0; // Select the correct channel according to the
scanned network interface list
18     if (!ec_master_->init(channel_index)) {
19         std::cout << "Initialization failed" << std::endl;
20         return -1;
21     }
22     std::cout << "Connection Success" << std::endl;
23     // EtherCAT master enters OP state
24     if (!ec_master_->start()) {
25         std::cout << "Failed to start device" << std::endl;
26         return -1;
27     }
```

```

28 // EtherCAT master starts background data exchange communication (non-
blocking)
29 ec_master_->run();
30
31 /***** Setting up RPDO sending callback and TPDO processing *****/
32 // EtherCAT sending function
33 auto send_func = [ec_master_](const unsigned char* data, unsigned int size) {
34     return ec_master_->setOutputs(data, size);
35 };
36 // Wrapping with std::function
37 std::function<bool(const unsigned char*, unsigned int)> func = send_func;
38
39 // Setting EtherCAT data sending callback
40 lhp_lib_->set_send_rpdo_callback_ex(&func);
41
42 // Creating a monitoring thread to parse TPDO data through the interface
43 std::atomic<bool> stop{false};
44 std::thread t_monitor([=, &stop]() {
45     while (!stop) {
46         // Continuously obtaining TPDO data and passing it to the SDK for parsing
47         int input_size = ec_master_->getInputSize();
48         std::vector<uint8_t> in_buffer(input_size);
49         if (ec_master_->getInputs(in_buffer.data(), input_size)) {
50             lhp_lib_->set_tpdo_data_decode(in_buffer.data(), input_size);
51         }
52         std::this_thread::sleep_for(std::chrono::milliseconds(10));
53     }
54 });
55
56 // Initializing the library
57 if (lhp_lib_->initial(lhplib::LCN_ECATE) != lhplib::LER_NONE) {
58     std::cerr << "LHandProLib initialization failed!" << std::endl;
59     return -1;
60 }
61
62
63 /***** Motion interface calls *****/
64 // Set control mode
65 lhp_lib_->set_control_mode(0, lhplib::LCM_POSITION);
66 // Enable all
67 lhp_lib_->set_enable(0, 1);
68
69
70 // Obtain the current degree of freedom
71 int dof_total = 0, dof_active = 0;
72 lhp_lib_->get_dof(&dof_total, &dof_active);
73

```

```

74 // Target angle for each joint
75 std::vector<float> target_angles = {30.0f, 10.0f, 60.0f, 60.0f, 60.0f,
76 60.0f};
77 // Set motion parameters
78 for (int i = 0; i < dof_active; ++i) {
79     lhp_lib->set_target_angle(i + 1, target_angles[i]); // Set motion
parameters
80     lhp_lib->set_angular_velocity(i + 1, 200);           // Set speed (200
°/s)
81     lhp_lib->set_max_current(i + 1, 1000);              // Set maximum
current limit (1000%)
82 }
83 // Drive all motors to move
84 lhp_lib->move_motors();
85 // Waiting for the end of the exercise
86 std::this_thread::sleep_for(std::chrono::milliseconds(1000));
87
88 // Obtain the current angle
89 for (int i = 0; i < dof_active; ++i) {
90     float angle = 0;
91     if (lhp_lib->get_now_angle(i + 1, &angle) == lhplib::LER_NONE) {
92         std::cout << "Current joint: " << i + 1 << " Angle: " << angle << "°" <<
std::endl;
93     }
94 }
95
96 /***** Sensor interface calls *****/
97 // Read the pressure distribution at the tip of the thumb
98 int count = 0;
99 lhp_lib->get_finger_pressure(lhplib::LSS_FINGER_1_1, nullptr, &count);
100 std::vector<float> pressures(count);
101 lhp_lib->get_finger_pressure(lhplib::LSS_FINGER_1_1, pressures.data(),
&count);
102
103 for (int i = 0; i < count; ++i) {
104     std::cout << "Sensor: " << i << " pressure: " << pressures[i] <<
std::endl;
105 }
106
107 // Read normal force
108 float normal_force;
109 lhp_lib->get_finger_normal_force(lhplib::LSS_FINGER_1_1, &normal_force);
110 std::cout << "Sensor: " << lhplib::LSS_FINGER_1_1 << " Normal force: " <<
normal_force << std::endl;
111
112 // Read tangential force

```

```

113     float tangential_force;
114     lhp_lib->get_finger_tangential_force(lhplib::LSS_FINGER_1_1,
    &tangential_force);
115     std::cout << "Sensor: " << lhplib::LSS_FINGER_1_1 << " Tangential force: "
    << tangential_force << std::endl;
116
117     // Read the direction of tangential force
118     float force_direction;
119     lhp_lib->get_finger_force_direction(lhplib::LSS_FINGER_1_1,
    &force_direction);
120     std::cout << "Sensor: " << lhplib::LSS_FINGER_1_1 << " direction: " <<
    force_direction << std::endl;
121
122     // Read proximity
123     float proximity;
124     lhp_lib->get_finger_proximity(lhplib::LSS_FINGER_1_1, &proximity);
125     std::cout << "Sensor: " << lhplib::LSS_FINGER_1_1 << " proximity: " <<
    proximity << std::endl;
126
127     // Close
128     lhp_lib->close();
129
130     return 0;
131 }

```

Other examples

Please refer to the example projects for each communication version in the SDK development kit.

| LHandProLib-API-share-LHandProLib-examples

代码块

```

1  LHandProLib-API
2  |—— bin
3  |   |—— LHandProLib.dll
4  |   |—— LHandProLib.pdb
5  |   |—— LHandProLibd.dll
6  |   |—— LHandProLibd.pdb
7  |   |—— LHandProLib_EtherCAT_Test.exe
8  |—— include
9  |   |—— LHandProLib
10 |       |—— LHandProLib.hpp
11 |       |—— LHandProLib.h
12 |—— lib
13 |   |—— LHandProLib.lib

```

```
14 |   └─ LHandProLibd.lib
15 |   └─ share
16 |       └─ LHandProLib
17 |           └─ examples
18 |               └─ EtherCAT_cpp
19 |                   └─ EtherCAT_python
```

Frequently Asked Questions

Q: Why does `initial()` return `LER_KEY_FUNC_UNINIT` ?

A: The `set_recv_data_decode` callback function is not set.

Q: How to view communication data?

A: Use `log_on(true)` and call `log_send(nullptr, 0)` and `log_recv(nullptr, 0)` to print all data.

Q: Sensor data is not updating?

A: Ensure that `set_tpdo_data_decode` (EtherCAT) is called periodically.

Version Update History

- **v1.0:** Initial version, supporting EtherCAT, 6/15 DOF, tactile sensor reading, and logging system.
- **v1.1:** Added equivalent control interface and optimized the EtherCAT C++ sample program.
- **v1.2:** Provided C wrapper header files and EtherCAT Python sample programs, optimized some implementation logic, and enhanced stability.
- **v1.3:** Added CANFD communication support.
- **v1.4:** Improved documentation and fixed known issues.
- **v1.5:** Improved sensor interface.

Copyright Notice: © Leadtron All rights reserved.

User Agreement: Using this software indicates that you agree to the [User Agreement](#).

