

libsvah Library API Reference

v86

Peter Felecan

assisted by Claude (Anthropic), model Sonnet 5

September 5, 2026

Contents

1	Introduction	3
2	HalCan — CAN Hardware Abstraction Layer	3
2.1	Types	3
2.1.1	HalCanFrame	3
2.1.2	HalCanStatus	4
2.1.3	HalCanRxCallback	4
2.1.4	HAL_CAN_RX_BUFFER_SIZE	4
2.2	Functions	4
2.2.1	HalCanInit	4
2.2.2	HalCanSetRxCallback	5
2.2.3	HalCanTransmit	5
2.2.4	HalCanReceive	5
2.2.5	HalCanClose	5
3	HalTime — Monotonic Time Abstraction	6
3.1	Functions	6
3.1.1	HalTimeMs	6
3.1.2	HalTimeUs	6
4	CanopenSimple — CANSimple Protocol Codec	6
4.1	Types	6
4.1.1	CansimpCmd	6
4.1.2	AxisState	7
4.1.3	ControlMode	7
4.1.4	InputMode	7
4.2	Functions	8

4.2.1	CansimpMakeId	8
4.2.2	CansimpParseId	8
4.2.3	CansimpEncodeSetInputPos	8
4.2.4	CansimpEncodeSetInputVel	8
4.2.5	CansimpEncodeSetInputTorque	8
4.2.6	CansimpEncodeSetControllerMode	9
4.2.7	CansimpEncodeSetLimits	9
4.2.8	CansimpEncodeSetTrajVelLimit	9
4.2.9	CansimpEncodeSetTrajAccelLimit	9
4.2.10	CansimpEncodeSetTrajInertia	9
4.2.11	CansimpEncodeSetAbsolutePos	10
4.2.12	CansimpEncodeSetPosGain	10
4.2.13	CansimpEncodeSetVelGains	10
4.2.14	CansimpEncodeSetAxisState	10
4.2.15	CansimpDecodeHeartbeat	10
4.2.16	CansimpDecodeEncoderEstimates	11
4.2.17	CansimpDecodeEncoderCount	11
4.2.18	CansimpDecodeIq	11
4.2.19	CansimpDecodeTemperature	11
4.2.20	CansimpDecodeBusVoltageCurrent	11
5	OdriveApi — High-Level ODrive Interface	11
5.1	Types	12
5.1.1	OdriveStatus	12
5.1.2	OdriveAxis	12
5.2	Functions	12
5.2.1	OdriveAxisInit	12
5.2.2	OdriveSetAxisState	13
5.2.3	OdriveWaitHeartbeat	13
5.2.4	OdriveSetInputPos	13
5.2.5	OdriveSetInputVel	13
5.2.6	OdriveSetInputTorque	13
5.2.7	OdriveSetControllerMode	14
5.2.8	OdriveSetLimits	14
5.2.9	OdriveClearErrors	14
5.2.10	OdriveReboot	14
5.2.11	OdriveRequestEncoderEstimates	14
5.2.12	OdriveRequestIq	14
5.2.13	OdriveRequestTemperature	14
5.2.14	OdriveRequestBusVoltage	15
5.2.15	OdriveFeedWatchdog	15
5.2.16	OdriveProcessHeartbeat	15
5.2.17	OdriveProcessFrame	15

5.2.18	<code>SvahLogSyslogCallback</code>	15
5.2.19	<code>SvahLogStderrCallback</code>	16
5.3	Syslog usage example	16

List of Figures

List of Tables

1 Introduction

This document is the API reference for `libsvah`, the C library for controlling ODrive BLDC motor controllers via the CANSimple protocol.

The library is structured in three layers, each with its own header:

Header	Layer	Portability
<code>HalCan.h</code>	CAN hardware abstraction	Platform-specific
<code>HalTime.h</code>	Monotonic time abstraction	Platform-specific
<code>CanopenSimple.h</code>	CANSimple frame codec	Portable
<code>OdriveApi.h</code>	High-level ODrive commands	Portable

Platform-specific implementations currently provided:

File	Platform	Interface
<code>HalCanLinux.c</code>	Linux	SocketCAN
<code>HalTimeLinux.c</code>	Linux	CLOCK_MONOTONIC

2 HalCan — CAN Hardware Abstraction Layer

Defined in `HalCan.h`. Provides a platform-independent interface for CAN frame transmission and reception.

The receive model is **hybrid**: a callback function pointer (nullable) can be registered for interrupt-driven reception. When the callback is `NULL`, incoming frames are buffered in a ring buffer for retrieval by polling via `HalCanReceive()`.

2.1 Types

2.1.1 `HalCanFrame`

A single CAN 2.0B data frame.

```
typedef struct
{
    uint32_t id;          /* 11-bit or 29-bit CAN identifier */
    uint8_t dlc;          /* Data Length Code (0..8) */
    uint8_t data[8];      /* Payload bytes */
    uint8_t isExtended;   /* Non-zero if 29-bit extended frame */
    uint8_t isRemote;     /* Non-zero if RTR frame */
} HalCanFrame;
```

2.1.2 HalCanStatus

Return code for all HAL CAN functions.

Value	Meaning
HAL_CAN_OK	Success
HAL_CAN_NO_FRAME	No frame available (polling, non-fatal)
HAL_CAN_ERR_INIT	Initialisation failure
HAL_CAN_ERR_TX	Transmit failure
HAL_CAN_ERR_RX	Receive failure
HAL_CAN_ERR_PARAM	Invalid parameter (NULL pointer, etc.)

2.1.3 HalCanRxCallback

Prototype for the optional receive callback.

```
typedef void (*HalCanRxCallback)(const HalCanFrame *pFrame,
                                void                *pUserData);
```

The callback may be invoked from an interrupt context on bare-metal targets. Keep it short and non-blocking.

2.1.4 HAL_CAN_RX_BUFFER_SIZE

Compile-time constant controlling the receive ring buffer depth. Default: 16. Must be a power of two. Override with `-DHAL_CAN_RX_BUFFER_SIZE=n` at compile time.

2.2 Functions

2.2.1 HalCanInit

```
HalCanStatus HalCanInit(HalCanHandle **ppHandle,
                        const char    *pInterface,
                        uint32_t       pB bitrate);
```

Initialise the CAN interface.

ppHandle out — receives the allocated handle on success; set to **NULL** on failure.

pInterface platform-specific name. Linux: "can0", "vcan0", etc. ESP32: ignored (single TWAI peripheral).

pBtrrate bus bitrate in bits/s (e.g. 250000, 500000, 1000000). On Linux, the bitrate must be configured externally via `ip(8)`; this value is informational only.

Returns **HAL_CAN_OK** on success, **HAL_CAN_ERR_INIT** on failure.

2.2.2 HalCanSetRxCallback

```
void HalCanSetRxCallback(HalCanHandle    *pHandle,
                        HalCanRxCallback pCallback,
                        void              *pUserData);
```

Register an optional receive callback. Pass **NULL** to revert to polling mode. Call after **HalCanInit** and before frames are expected.

2.2.3 HalCanTransmit

```
HalCanStatus HalCanTransmit(HalCanHandle    *pHandle,
                           const HalCanFrame *pFrame);
```

Transmit a single CAN frame. Returns **HAL_CAN_OK** on success, **HAL_CAN_ERR_TX** on failure.

2.2.4 HalCanReceive

```
HalCanStatus HalCanReceive(HalCanHandle *pHandle,
                          HalCanFrame  *pFrame);
```

Retrieve one frame in polling mode.

Return value	Meaning
HAL_CAN_OK	Frame written to <i>*pFrame</i>
HAL_CAN_NO_FRAME	No frame available
HAL_CAN_ERR_RX	Receive error
HAL_CAN_ERR_PARAM	<i>pHandle</i> or <i>pFrame</i> is NULL

2.2.5 HalCanClose

```
void HalCanClose(HalCanHandle *pHandle);
```

Release all resources. After this call **pHandle** must not be used.

3 HalTime — Monotonic Time Abstraction

Defined in `HalTime.h`. Provides a platform-independent monotonic time source for timeout management and watchdog feeding.

Both functions return `uint32_t` values that **wrap at `UINT32_MAX`**. Always compute elapsed time with unsigned subtraction:

```
uint32_t lStart    = HalTimeMs();
uint32_t lElapsed = HalTimeMs() - lStart; /* correct across wrap */
```

3.1 Functions

3.1.1 HalTimeMs

```
uint32_t HalTimeMs(void);
```

Monotonic timestamp in milliseconds. Wraps at `UINT32_MAX` (~49.7 days).

3.1.2 HalTimeUs

```
uint32_t HalTimeUs(void);
```

Monotonic timestamp in microseconds. Wraps at `UINT32_MAX` (~71.6 min). Use for short-duration measurements only.

4 CanopenSimple — CANSimple Protocol Codec

Defined in `CanopenSimple.h`. Portable, platform-independent encoding and decoding of ODrive CANSimple protocol frames.

The CANSimple message identifier is:

```
msg_id = (axis_id << 5) | ~cmd_id~
```

Payload fields are encoded by `memcpy` of native C types (`float`, `uint32_t`, `int16_t`) into an 8-byte payload, in host byte order (little-endian on x86/ARM/ESP32).

4.1 Types

4.1.1 CansimpCmd

Enumeration of all CANSimple command identifiers (5-bit `cmd_id` field). Notable values:

Constant	Value	Direction
CANSIMP_CMD_HEARTBEAT	0x001	RX
CANSIMP_CMD_SET_AXIS_STATE	0x007	TX
CANSIMP_CMD_SET_INPUT_POS	0x00C	TX
CANSIMP_CMD_SET_INPUT_VEL	0x00D	TX
CANSIMP_CMD_SET_INPUT_TORQUE	0x00E	TX
CANSIMP_CMD_CLEAR_ERRORS	0x018	TX

See `CanopenSimple.h` for the full enumeration.

4.1.2 AxisState

ODrive axis state machine states. Notable values:

Constant	Value
AXIS_STATE_IDLE	1
AXIS_STATE_FULL_CALIBRATION_SEQ	3
AXIS_STATE_CLOSED_LOOP_CONTROL	8

4.1.3 ControlMode

Controller operating mode.

Constant	Value
CONTROL_MODE_VOLTAGE_CONTROL	0
CONTROL_MODE_TORQUE_CONTROL	1
CONTROL_MODE_VELOCITY_CONTROL	2
CONTROL_MODE_POSITION_CONTROL	3

4.1.4 InputMode

How setpoints are interpreted by the controller.

Constant	Value	Description
INPUT_MODE_INACTIVE	0	No input
INPUT_MODE_PASSTHROUGH	1	Direct setpoint
INPUT_MODE_VEL_RAMP	2	Velocity ramp
INPUT_MODE_POS_FILTER	3	Second-order position filter
INPUT_MODE_TRAP_TRAJ	5	Trapezoidal trajectory planner
INPUT_MODE_TORQUE_RAMP	6	Torque ramp

4.2 Functions

4.2.1 CansimpMakeId

```
uint32_t CansimpMakeId(uint8_t pAxisId, CansimpCmd pCmdId);
```

Build a CANSimple 11-bit message identifier.

4.2.2 CansimpParseId

```
void CansimpParseId(uint32_t pMsgId,
                    uint8_t *pAxisId,
                    CansimpCmd *pCmdId);
```

Decompose a received CAN message identifier into axis id and command.

4.2.3 CansimpEncodeSetInputPos

```
void CansimpEncodeSetInputPos(uint8_t pAxisId,
                              float pInputPos,
                              int16_t pVelFeedForward,
                              int16_t pTorqueFeedForward,
                              HalCanFrame *pFrame);
```

Encode a Set Input Position command (cmd 0x00C).

pInputPos position setpoint in turns.

pVelFeedForward velocity feed-forward in turns/s, Q16.

pTorqueFeedForward torque feed-forward in Nm, Q16.

4.2.4 CansimpEncodeSetInputVel

```
void CansimpEncodeSetInputVel(uint8_t pAxisId,
                              float pInputVel,
                              float pTorqueFeedForward,
                              HalCanFrame *pFrame);
```

Encode a Set Input Velocity command (cmd 0x00D).

4.2.5 CansimpEncodeSetInputTorque

```
void CansimpEncodeSetInputTorque(uint8_t pAxisId,
                                  float pInputTorque,
                                  HalCanFrame *pFrame);
```

Encode a Set Input Torque command (cmd 0x00E).

pInputTorque torque setpoint in Nm.

4.2.6 CansimpEncodeSetControllerMode

```
void CansimpEncodeSetControllerMode(uint8_t      pAxisId,
                                     ControlMode  pControlMode,
                                     InputMode     pInputMode,
                                     HalCanFrame *pFrame);
```

Encode a Set Controller Mode command (cmd 0x00B). Must be sent before entering AXIS_STATE_CLOSED_LOOP_CONTROL.

4.2.7 CansimpEncodeSetLimits

```
void CansimpEncodeSetLimits(uint8_t      pAxisId,
                             float        pVelocityLimit,
                             float        pCurrentLimit,
                             HalCanFrame *pFrame);
```

Encode a Set Limits command (cmd 0x00F).

pVelocityLimit velocity limit in turns/s.

pCurrentLimit current limit in A.

4.2.8 CansimpEncodeSetTrajVelLimit

```
void CansimpEncodeSetTrajVelLimit(uint8_t      pAxisId,
                                   float        pTrajVelLimit,
                                   HalCanFrame *pFrame);
```

Encode a Set Traj Vel Limit command (cmd 0x011). Used with INPUT_MODE_TRAP_TRAJ.

4.2.9 CansimpEncodeSetTrajAccelLimit

```
void CansimpEncodeSetTrajAccelLimit(uint8_t      pAxisId,
                                     float        pTrajAccelLimit,
                                     float        pTrajDecelLimit,
                                     HalCanFrame *pFrame);
```

Encode a Set Traj Accel Limit command (cmd 0x012). Both values in turns/s². pTrajDecelLimit should be positive.

4.2.10 CansimpEncodeSetTrajInertia

```
void CansimpEncodeSetTrajInertia(uint8_t      pAxisId,
                                  float        pInertia,
                                  HalCanFrame *pFrame);
```

Encode a Set Traj Inertia command (cmd 0x013). Units: Nm/(turns/s²).

4.2.11 CansimpEncodeSetAbsolutePos

```
void CansimpEncodeSetAbsolutePos(uint8_t      pAxisId,
                                float         pAbsPos,
                                HalCanFrame *pFrame);
```

Encode a Set Absolute Position command (cmd 0x019). Sets the encoder shadow count without commanding motion.

4.2.12 CansimpEncodeSetPosGain

```
void CansimpEncodeSetPosGain(uint8_t      pAxisId,
                             float         pPosGain,
                             HalCanFrame *pFrame);
```

Encode a Set Pos Gain command (cmd 0x01A). Units: (turns/s)/turn.

4.2.13 CansimpEncodeSetVelGains

```
void CansimpEncodeSetVelGains(uint8_t      pAxisId,
                              float         pVelGain,
                              float         pVelIntegratorGain,
                              HalCanFrame *pFrame);
```

Encode a Set Vel Gains command (cmd 0x01B).

pVelGain velocity proportional gain in Nm/(turns/s).

pVelIntegratorGain velocity integrator gain in Nm/turn.

4.2.14 CansimpEncodeSetAxisState

```
void CansimpEncodeSetAxisState(uint8_t      pAxisId,
                               AxisState     pState,
                               HalCanFrame *pFrame);
```

Encode a Set Axis State command (cmd 0x007).

4.2.15 CansimpDecodeHeartbeat

```
int CansimpDecodeHeartbeat(const HalCanFrame *pFrame,
                           uint32_t         *pAxisError,
                           AxisState         *pAxisState,
                           uint8_t          *pProcedureResult,
                           uint8_t          *pTrajDone);
```

Decode a Heartbeat frame (cmd 0x001). Returns 0 on success, -1 if the frame is not a valid heartbeat. Any output pointer may be NULL if the corresponding field is not needed.

4.2.16 CansimpDecodeEncoderEstimates

```
int CansimpDecodeEncoderEstimates(const HalCanFrame *pFrame,  
                                float               *pPosEstimate,  
                                float               *pVelEstimate);
```

Decode a `Get_Encoder_Estimates` frame (cmd 0x009). Returns 0 on success, -1 on error.

4.2.17 CansimpDecodeEncoderCount

```
int CansimpDecodeEncoderCount(const HalCanFrame *pFrame,  
                             int32_t          *pShadowCount,  
                             int32_t          *pCountCpr);
```

Decode a `Get_Encoder_Count` frame (cmd 0x00A).

4.2.18 CansimpDecodeIq

```
int CansimpDecodeIq(const HalCanFrame *pFrame,  
                   float              *pIqSetpoint,  
                   float              *pIqMeasured);
```

Decode a `Get_Iq` frame (cmd 0x014). Units: A.

4.2.19 CansimpDecodeTemperature

```
int CansimpDecodeTemperature(const HalCanFrame *pFrame,  
                             float            *pFetTemp,  
                             float            *pMotorTemp);
```

Decode a `Get_Temperature` frame (cmd 0x015). Units: °C.

4.2.20 CansimpDecodeBusVoltageCurrent

```
int CansimpDecodeBusVoltageCurrent(const HalCanFrame *pFrame,  
                                  float              *pBusVoltage,  
                                  float              *pBusCurrent);
```

Decode a `Get_Bus_Voltage_Current` frame (cmd 0x017). Units: V and A respectively.

5 OdriveApi — High-Level ODrive Interface

Defined in `OdriveApi.h`. Builds on `CanopenSimple` and `HalCan` to provide application-level motor control commands.

Initialise an `OdriveAxis` context. `pAxis` must be pre-allocated (stack or static). The HAL handle is not owned and must remain valid for the lifetime of `pAxis`. All feedback fields are zeroed.

5.2.2 `OdriveSetAxisState`

```
OdriveStatus OdriveSetAxisState(OdriveAxis *pAxis,
                                AxisState   pState,
                                uint32_t    pTimeoutMs);
```

Request a state transition and wait for acknowledgement via heartbeat polling. Returns `ODRIVE_ERR_TIMEOUT` if the axis does not report the requested state within `pTimeoutMs` milliseconds.

5.2.3 `OdriveWaitHeartbeat`

```
OdriveStatus OdriveWaitHeartbeat(OdriveAxis *pAxis,
                                  uint32_t    pTimeoutMs);
```

Block until one heartbeat frame is received, updating `pAxis->lastState` and `pAxis->lastError` on success. Returns `ODRIVE_ERR_TIMEOUT` if no heartbeat arrives within `pTimeoutMs` milliseconds — for example because the CAN bus was disconnected. Used by `svah --hold` to detect bus disruption after the initial state transition; see TestPlan.org, PI-05.

5.2.4 `OdriveSetInputPos`

```
OdriveStatus OdriveSetInputPos(OdriveAxis *pAxis,
                                float       pInputPos,
                                int16_t     pVelFeedForward,
                                int16_t     pTorqueFeedForward);
```

Send a position setpoint. The axis must be in `AXIS_STATE_CLOSED_LOOP_CONTROL`.

5.2.5 `OdriveSetInputVel`

```
OdriveStatus OdriveSetInputVel(OdriveAxis *pAxis,
                                float       pInputVel,
                                float       pTorqueFeedForward);
```

Send a velocity setpoint.

5.2.6 `OdriveSetInputTorque`

```
OdriveStatus OdriveSetInputTorque(OdriveAxis *pAxis,
                                   float       pInputTorque);
```

Send a torque setpoint in Nm.

5.2.7 OdriveSetControllerMode

```

OdriveStatus OdriveSetControllerMode(OdriveAxis *pAxis,
                                     ControlMode  pControlMode,
                                     InputMode     pInputMode);

```

Set the control mode and input mode. Typically called before entering `AXIS_STATE_CLOSED_LOOP_CONTROL`.

5.2.8 OdriveSetLimits

```

OdriveStatus OdriveSetLimits(OdriveAxis *pAxis,
                             float       pVelocityLimit,
                             float       pCurrentLimit);

```

Set velocity limit (turns/s) and current limit (A).

5.2.9 OdriveClearErrors

```

OdriveStatus OdriveClearErrors(OdriveAxis *pAxis);

```

Send the Clear Errors command (cmd 0x018).

5.2.10 OdriveReboot

```

OdriveStatus OdriveReboot(OdriveAxis *pAxis);

```

Send the Reboot command. The axis context becomes invalid after this call; allow time for the ODrive to restart before re-initialising.

5.2.11 OdriveRequestEncoderEstimates

```

OdriveStatus OdriveRequestEncoderEstimates(OdriveAxis *pAxis);

```

Send an RTR frame requesting encoder estimates. The response updates `pAxis->posEstimate` and `pAxis->velEstimate` when processed by `OdriveProcessFrame`.

5.2.12 OdriveRequestIq

```

OdriveStatus OdriveRequestIq(OdriveAxis *pAxis);

```

Request Iq feedback. Response updates `iqSetpoint` and `iqMeasured`.

5.2.13 OdriveRequestTemperature

```

OdriveStatus OdriveRequestTemperature(OdriveAxis *pAxis);

```

Request temperature feedback. Response updates `fetTemp` and `motorTemp`.

5.2.14 OdriveRequestBusVoltage

```
OdriVeStatus OdriVeRequestBusVoltage(OdriVeAxis *pAxis);
```

Request bus voltage/current feedback. Response updates `busVoltage` and `busCurrent`.

5.2.15 OdriveFeedWatchdog

```
OdriVeStatus OdriVeFeedWatchdog(OdriVeAxis *pAxis);
```

Send a lightweight RTR frame to reset the ODrive watchdog timer. Call at a rate faster than `axis.config.watchdog_timeout` (configured on the ODrive). Sends a `Get_Encoder_Estimates` RTR, which also produces a useful response.

5.2.16 OdriveProcessHeartbeat

```
int OdriVeProcessHeartbeat(OdriVeAxis      *pAxis,
                           const HalCanFrame *pFrame);
```

Update `lastState`, `lastError`, and `trajDone` from a Heartbeat frame. Returns `ODRIVE_OK` if processed, 1 if ignored, -1 on error.

5.2.17 OdriveProcessFrame

```
int OdriVeProcessFrame(OdriVeAxis      *pAxis,
                       const HalCanFrame *pFrame);
```

Dispatch a received frame to the appropriate decoder. Handles `Heartbeat`, `Get_Encoder_Estimates`, `Get_Encoder_Count`, `Get_Iq`, `Get_Temperature`, and `Get_Bus_Voltage_Current`. Updates the corresponding `OdriVeAxis` fields. Returns `ODRIVE_OK` if processed, 1 if ignored (different axis or unknown `cmd_id`), -1 on decode error.

Preferred over `OdriVeProcessHeartbeat` when multiple message types are expected.

5.2.18 SvahLogSyslogCallback

Ready-made callback for the Linux system logger.

```
void SvahLogSyslogCallback(SvahLogLevel pLevel, const char *pMessage);
```

Routes messages to `syslog(3)`. Level mapping:

<u>SvahLogLevel</u>	<u>syslog priority</u>
Continued on next page	

Continued from previous page	
SvahLogLevel	syslog priority
SVAH_LOG_DEBUG	LOG_DEBUG
SVAH_LOG_INFO	LOG_INFO
SVAH_LOG_WARN	LOG_WARNING
SVAH_LOG_ERROR	LOG_ERR

The caller must call `openlog(3)` before registering this callback if a custom ident or facility is required.

5.2.19 SvahLogStderrCallback

Ready-made callback for standard error output.

```
void SvahLogStderrCallback(SvahLogLevel pLevel, const char *pMessage);
```

Writes [svah LEVEL] message\n to stderr.

5.3 Syslog usage example

This example shows the recommended logging setup for a Linux/SocketCAN application using libsvah.

```
/*
 * Recommended logging setup for Linux.
 *
 * Default: route library messages to the system logger.
 * Override: --log-on-stderr sends them to stderr instead.
 *
 * The log level is controlled by --debug (DEBUG), normal (INFO),
 * or --quiet (WARN).
 *
 * Include rationale:
 *   syslog.h : openlog(3), LOG_PID, LOG_CONS, LOG_USER
 */
#include <syslog.h>
#include "SvahLog.h"
#include "OdriveApi.h"

int main(int argc, char *argv[])
{
    int lLogOnStderr = 0; /* set from --log-on-stderr option */
    int lDebug       = 0; /* set from --debug option          */
    int lQuiet       = 0; /* set from --quiet option         */
```

```

SvahLogLevel lLevel = lDebug ? SVAH_LOG_DEBUG :
                      lQuiet ? SVAH_LOG_WARN  :
                      SVAH_LOG_INFO;

if(lLogOnStderr)
{
    /* Redirect to stderr – useful during development and debugging */
    SvahLogSetCallback(SvahLogStderrCallback, lLevel);
}
else
{
    /* Default: log to syslog.
     * Entries appear in /var/log/syslog (or journalctl on systemd).
     * LOG_PID: include the process ID in each entry.
     * LOG_CONS: fall back to the console if syslog is unavailable.
     * LOG_USER: general user-level messages. */
    openlog("svah", LOG_PID | LOG_CONS, LOG_USER);
    SvahLogSetCallback(SvahLogSyslogCallback, lLevel);
}

/* All subsequent library calls route their diagnostics through
 * the registered callback. The application never calls
 * SvahLogWrite directly. */
HalCanHandle *lHandle = NULL;
HalCanInit(&lHandle, "can0", 250000);
/* ... rest of application ... */

if(!lLogOnStderr)
{
    closelog();
}
return 0;
}

```

Observing syslog output:

```

# On systemd systems:
journalctl -f -t svah

```

```

# On traditional syslog:
tail -f /var/log/syslog | grep svah

```

```
# During development – bypass syslog entirely:
svah --log-on-stderr --debug --interface vcan0 --no-motion
```

Bare-metal usage:

On bare-metal targets (ESP32, SAMD) where neither `syslog` nor `stderr` exists, the application registers its own callback:

```
static void UartLog(SvahLogLevel pLevel, const char *pMsg)
{
    uart_puts("[");
    uart_puts(SvahLogLevelName(pLevel));
    uart_puts("] ");
    uart_puts(pMsg);
    uart_puts("\r\n");
}

/* In application init – before any libsvah call: */
SvahLogSetCallback(UartLog, SVAH_LOG_WARN);
```