

Carina VIO SDK — A1088 API Reference

Version: v3.4.16

Updated: 2026-09-17

Header: api/carina_a1088.h (product identification also available in api/carina_product.h)

Library: libcarina_vio.so

Table of Contents

1. Overview

2. Quick Start

3. Data Structures

4. Callbacks

5. Core API

6. Configuration

7. Examples

8. Thread Safety & Data Lifetime

9. Calibration C API

10. carina_vio C++ API (feeding external image/IMU)

11. carina.h (primary C++ API overview)

1. Overview

C API for the A1088 VIO tracking device. Pure C interface, compatible with C and C++.

Device Variants

Type	device_type	Description
Single-chip	10	Stereo/quad, auto exposure, VIO tracking
Dual-chip	20	Quad camera, dual-chip combined output
ISP	30	Stereo, integrated ISP
Monocular module	12	MD_FT70, monocular
Data-cap glasses	19	Quad + dual RGB (RK3588 / Linux aarch64 only , see § 5.14; identification see § 5.13)

Typical Lifecycle



Removed APIs

These old APIs have been removed:

Removed	Replacement
<code>carina_a1088_use_config</code>	Config passed via <code>carina_a1088_init</code>
<code>carina_a1088_trigger_cam_stop</code>	No manual trigger needed
<code>carina_a1088_trigger_cam_start</code>	No manual trigger needed
<code>carina_a1088_set_daisch_imu_pms</code>	Handled by calibration flow
<code>carina_a1088_get_daisch_imu_sn</code>	Handled by calibration flow
<code>carina_a1088_get_dev_info</code>	Handled by calibration flow
<code>carina_a1088_stm_get_device_version</code>	Handled automatically
<code>carina_a1088_stm_get_cpld_version</code>	Handled automatically
<code>carina_a1088_stm_suspend</code>	Handled automatically
<code>carina_a1088_stm_resume</code>	Handled automatically

2. Quick Start

```
#include "carina_a1088.h"
#include "carina_error.h"

// 1. Config as YAML string
const char* config = "device_type: 10\nmax_cameras: 2\nCamera.fps: 30\n...";

// 2. Initialize
int ret = carina_a1088_init(
    (char*)config,          // YAML config (required)
    "./database.bin",       // ORB vocabulary (required)
    -1, -1, -1              // USB params: -1 = auto-detect
);
if (ret != 0) {
    printf("Init failed: %d\n", ret);
    return -1;
}

// 3. Start VIO
ret = carina_a1088_start(
    my_pose_callback,       // pose callback
    NULL,                   // vsync (optional)
    my_imu_callback,        // imu callback
    my_image_callback,      // image callback
    NULL,                   // feature points (optional)
    NULL,                   // events (optional)
    NULL                    // user data
);
if (ret != 0) { /* handle error */ }

// 4. Stop & release
carina_a1088_stop();
carina_a1088_release();
```

Tip: For dual-chip devices, use `carina_double_a1088_init` instead of `carina_a1088_init`. Everything else stays the same.

3. Data Structures

3.1 Device Mode

```
struct carina_device_mode {
    uint8_t cam_en;         // Camera enable mask
    uint8_t data_fps;       // Data frame rate
    uint8_t cam_fps;        // Camera frame rate
    uint32_t imu_fps;       // IMU frame rate
};
```

3.2 Feature Points

```
struct carina_orb_point {
    int octave;           // Image pyramid level
    float angle;          // Feature orientation (degrees)
    float response;       // Feature strength
    float x, y;           // Image coordinates
    unsigned char desc[32]; // ORB descriptor (256 bits)
    unsigned int id;       // Unique ID
};

struct carina_lk_point {
    float x, y;           // Image coordinates
    unsigned int id;       // Unique ID
};

struct carina_points {
    struct carina_lk_point **points_lk;    // LK points per camera
    struct carina_orb_point **points_orb;  // ORB points per camera
    int points_lk_rows;                    // Cameras with LK points
    int points_lk_cols[4];                 // LK count per camera
    int points_orb_rows;                   // Cameras with ORB points
    int points_orb_cols[4];                // ORB count per camera
};
```

3.3 Nordic Wireless Data (AR Glasses)

```
struct carina_nordic_pose_data {
    double timestamp;
    float fpose[16];    // 4x4 transformation matrix
    bool bsync;
};

struct carina_nordic_imu_data {
    double timestamp;
    float acc_data[3];  // acceleration
    float gyro_data[3]; // gyroscope
    bool bsync;
};
```

4. Callbacks

All callbacks run on **internal worker threads**. Pointers are valid **only during the callback** (see § 8).

```

// Pose – 32 floats
// pose[0..15] = TWB transformation matrix (4×4, column-major)
// pose[16..18] = velocity (vx, vy, vz)
// pose[22..24] = gyro bias
// pose[25..27] = accel bias
// pose[28] = static ratio
// pose[30] = tracking state
typedef void (*CarinaA1088PoseCallBackType)(float *pose, double timestamp, void *user_data);

// VSync
typedef void (*CarinaA1088VsyncCallBackType)(double timestamp, void *user_data);

// IMU – 6 floats
// data[0..2] = acceleration (m/s²)
// data[3..5] = gyroscope (rad/s)
typedef void (*CarinaA1088ImuCallBackType)(float *imu_data, double timestamp, void *user_data);

// Image – up to 4 grayscale images; NULL for unavailable cameras
typedef void (*CarinaA1088CameraCallBackType)(
    char *img0, char *img1, char *img2, char *img3,
    double timestamp, int w, int h, void *user_data
);

// Feature points (pass NULL if not needed)
typedef void (*CarinaA1088PointsCallBackType)(
    struct carina_points *points, double timestamp, void *user_data
);

// Events (pass NULL if not needed)
typedef void (*CarinaA1088EventCallBackType)(unsigned char event, void *user_data);

// HMD suspend/wakeup: 0=suspend, 1=wakeup
typedef void (*CarinaA1088SuspendCallBackType)(int state, void *user_data);

// Firmware update progress (0.0 ~ 1.0)
typedef void (*CarinaA1088ProcessCallBackType)(float progress, void *user_data);

// Nordic wireless data receive
typedef void (*CarinaA1088NordicDataCallBackType)(
    const char* data_val, int data_len,
    unsigned char nordic_trans_type, void* user_data
);

// UVC camera frame (MJPEG/NV12)
typedef void (*CarinaUvcFrameCallback)(
    const uint8_t* data, int size,
    int width, int height, int64_t timestamp, void* user_data
);

```

5. Core API

5.1 Lifecycle

```
// ————— Init —————

// Single-chip / stereo
// custom_config: YAML config string (required)
// vocab_file_path: ORB vocabulary (database.bin)
// fd/bus/addr: USB params, -1 = auto-detect
int carina_al088_init(char *custom_config, char *vocab_file_path,
                    int fd = -1, int bus = -1, int addr = -1);

// Dual-chip / quad
int carina_double_al088_init(char *custom_config, char *vocab_file_path,
                            int fd_main = -1, int bus_main = -1, int addr_main = -1,
                            int fd_div = -1, int bus_div = -1, int addr_div = -1);

// ————— Start / Stop —————

int carina_al088_start(/* callbacks */); // start VIO
int carina_al088_stop();                // stop
int carina_al088_release();              // release all resources
int carina_al088_pause();                // pause (preserve state)
int carina_al088_resume();               // resume

// ————— HMD callback —————

int carina_al088_set_hmd_state_callback(
    CarinaAl088SuspendCallBackType cb, void* user_data);
```

5.2 Device Info

```
char *carina_al088_get_sn();              // serial number
char *carina_al088_get_cam_param();       // camera calibration YAML
char *carina_al088_get_config();          // current config YAML
char *carina_al088_get_config_des(pass);  // decrypted config
char *carina_al088_get_sdk_version();     // SDK version string
char *carina_al088_get_firmware_version(); // firmware version string
bool carina_al088_is_device_connect();    // is device connected?
bool carina_al088_get_cur_status();       // is device operational?
int32_t carina_al088_get_device_type();   // device type enum value
```

char* return values point to **internal static buffers**. Do NOT free them.
Subsequent calls overwrite the buffer — copy data if you need it to persist.

5.3 Pose

```
// pose[16] = 4x4 transformation matrix (column-major), IMU-predicted
int carina_al088_get_imu_pose(float *pose, double predicttime);

// pose[32] = full state (velocity, bias), for OpenGL rendering
int carina_al088_get_gl_pose(float *pose, double predicttime);
```

5.4 Device Control

```
// Display
int carina_al088_switch_display_mode(unsigned char mode); // 0=2D, 1=3D
int carina_al088_set_display_level(unsigned char level); // brightness 0~4
int carina_al088_get_display_level(); // get brightness
int carina_al088_set_led_color(unsigned char color); // LED color
bool carina_al088_detect_worn(); // wear detection
uint16_t carina_al088_get_proximity_val(); // proximity sensor
uint16_t carina_al088_get_key_status(); // key/touch state

// Sensor
int carina_al088_reset_pose(); // reset pose to origin
bool carina_al088_check_loop_state(); // loop closure state
int carina_al088_set_sync_ts(bool enable); // timestamp sync
void carina_al088_set_log_level(unsigned int level); // 0=off ~ 4=debug
int carina_al088_set_low_power_mode(bool enable); // low power mode
int carina_al088_set_static_check_time(float sec); // static detection time

// Runtime config update
bool carina_al088_update_custom_config(const char* config_val);
```

5.5 Exposure / Gain Control

```
// Set exposure or gain for specific camera(s)
// mode: 0x01=exposure, 0x02=gain
// cam_id: 0x01=CAM0, 0x02=CAM1, 0x04=CAM2, 0x08=CAM3
int carina_al088_set_exposure_gain_with_id(
    unsigned char mode, unsigned char cam_id, int value);

// Auto exposure
int carina_al088_open_auto_exposure(int minG, int maxG); // enable
int carina_al088_close_auto_exposure(); // disable
int carina_al088_open_auto_exposure_v2(int minG, int maxG); // 4-cam independent

// Read
float carina_al088_get_exposure_time(unsigned char camid); // ms
int carina_al088_get_exposure_gain(unsigned char camid); // gain multiplier
int carina_al088_get_auto_exp_status(); // 0=fixed,1=auto,2=indep
```

5.6 USB Custom Data

```
// Send (USB2.0: max 512 bytes, USB3.0: max 1024 bytes)
int carina_a1088_send_custom_data(const char *data, int len);

// Read (returns static buffer, min 32 bytes)
char* carina_a1088_read_custom_data(int len);
```

5.7 UVC Camera

Fixed VID:PID = 2bc5:052b. MJPEG on Win/Linux, NV12 on macOS.

```
int carina_a1088_uvc_open(int width, int height, int fps, int fd = -1);
void carina_a1088_uvc_close();
int carina_a1088_uvc_start(CarinaUvcFrameCallback cb, void* user_data);
void carina_a1088_uvc_stop(); // stop stream, device stays open
```

5.8 Nordic Wireless (AR Glasses)

```
bool carina_a1088_nordic_send_imu_data(
    const struct carina_nordic_imu_data* data, int size,
    unsigned char datatype, int level);

bool carina_a1088_nordic_send_pose_data(
    const struct carina_nordic_pose_data* data, int size,
    unsigned char datatype, int level);

void carina_a1088_nordic_data_callback(
    CarinaA1088NordicDataCallBackType cb,
    int trx_mode, void* user_data);
// trx_mode: 0x01=TX, 0x10=RX, 0x11=TRX

float carina_a1088_nordic_time_diff(); // TX-RX time diff (seconds)
```

5.9 Firmware Update

```
void carina_a1088_set_process_callback(
    CarinaA1088ProcessCallBackType cb, void* user_data);

bool carina_a1088_update_chip_fw(
    const char* fw_path, int vid, int pid, int fd);
```

5.10 Factory / Calibration Helpers

```
int carina_a1088_set_sn(const char* sn); // returns 1=ok 0=fail
int carina_a1088_set_dev_cali_info(const char* data, // returns 1=ok 0=fail
    int len, int mode);
```

5.11 Custom Parameter Storage

```
bool carina_al088_write_cus_para(const char* data, int len);
uint32_t carina_al088_read_cus_para(char** out_data);
```

5.12 Debug Helpers

```
// Get data/analysis thread IDs (debugging)
void carina_al088_get_thread_id(uint64_t* data_thread, uint64_t* analysis_thread);

// Map points as JSON (SLAM world frame, Z-up; for 3D visualisation)
// returns bytes written; -1 when the buffer is too small
int carina_al088_get_map_points_json(char* buf, int cap);

// Pipeline profiler as JSON (per-stage count/avg/p50/p95/p99/last, plus *.fps rows)
int carina_al088_get_pipeline_profiler_json(char* buf, int cap);

// Frame-rate tick: call once per frame; the profiler resolves fps once a second into that stage.
// Stage naming: cam0..cam3.fps / imu.fps / pose.fps / uvc.fps / rgb0.fps / rgb1.fps
void carina_al088_profiler_tick_fps(const char* stage);
```

5.13 Product Identification & Pre-open Probing

`device_type` maps to products **many-to-one**: **FT62 / FT63 / YuYu AR glasses share `device_type` 10**, while other products have their own type. The APIs below let you identify the product **before opening the device**.

```

// Product table (JSON array) – single source of truth for console / web console
// fields: id / device_type / is_double / cameras / name / name_long / has_uvc / has_dp
// Order == UI drop-down order. Internally owned read-only string – do NOT free.
const char* carina_a1088_get_product_table_json(void);

// After a successful init(): check whether the selected product matches the detected device_type
// out_detected_type / out_resolved_product_id may be NULL
// returns 0 = compatible, 1 = incompatible, -1 = not connected / bad args
int carina_a1088_check_product_match(int selected_product_id,
                                     int32_t* out_detected_type,
                                     int32_t* out_resolved_product_id);

// — The 4 signals below are available BEFORE opening the device:
//   pure enumeration, no open / no claim, no device state change —

// UVC camera present (VID:PID = 2bc5:052b); only FT63 / YuYu AR glasses have it
bool carina_a1088_uvc_present(void);

// Number of enumerated A1088 USB devices (0906:5678 main / 0906:5679 secondary)
// 2 = FT66 (dual-chip); 1 = single-chip product; -1 = enumeration unavailable
int carina_a1088_count_a1088_usb(void);

// Is a 3840x1200 display attached (via Linux DRM sysfs); unique to YuYu AR glasses
bool carina_a1088_dp_display_present(void);

// Both MIPI camera nodes of the data-cap glasses (/dev/video22, /dev/video31) present
bool carina_a1088_datacap_cameras_present(void);

// Aggregated pre-open verdict (JSON); internally owned read-only string – do NOT free
const char* carina_a1088_probe_preopen_json(void);

```

Product table (kProducts[] in `carina_product.h`, order == UI drop-down order):

id	Product	device_type	Cameras	UVC	DP
0	FT62	10	2	✗	✗
1	FT63	10	2	✓	✗
2	FT66	20	4 (dual-chip)	✗	✗
3	FT68	30	2	✗	✗
4	YuYu AR glasses	10	2	✓	✓
5	Data-cap glasses	19	4 + dual RGB	✗	✗
6	MD_FT70 monocular	12	1	✗	✗

`carina_a1088_probe_preopen_json()` returns:

Field	Type	Meaning
a1088_count	int	Enumerated A1088 count; -1 = undetermined
uvc	bool	UVC camera detected
dp_display	bool	3840x1200 display attached
datacap_cams	bool	Both data-cap MIPI nodes present
resolved_product_id	int	Uniquely resolved product id, else -1
candidates	int[]	Candidate ids when not unique
device_present	bool	At least one A1088 enumerated

Resolution priority (strongest exclusivity first):

Condition	Verdict
2 A1088 enumerated	FT66 (dual-chip)
3840x1200 display attached	YuYu AR glasses (only product with DP)
both /dev/video22 + /dev/video31	Data-cap glasses
UVC present	FT63
exactly 1 A1088	not unique — candidates {FT62, FT68, MD_FT70}
none enumerated	no device (or enumeration unavailable)

The standalone `carina_product.h` (also shipped in the SDK `include/`) can be used directly:

```
const CarinaProductDesc* carina_product_table(int* out_count);
const CarinaProductDesc* carina_product_get(int id);
bool carina_product_available(int id);
CarinaProductId carina_product_resolve(int device_type, bool has_uvc, bool has_dp);
CarinaProductId carina_product_resolve_preopen(const CarinaPreopenSignals& sig,
                                              int* out_candidates, int cap, int* out_count);
bool carina_product_type_compatible(int selected_id, int detected_device_type);
```

Platform filtering: `carina_product_table()` returns a **platform-filtered** table — the data-cap glasses only appear on Linux aarch64 (RK3588); Windows / macOS / x86 Linux will not list them. On Windows the four signals ``uvc`` / ``dp_display`` / ``a1088_count`` / ``datacap_cams`` always report "undetermined" (no libusb headers, no DRM sysfs, no `/dev/videoN`); resolution then relies on the remaining signals.

```
// Is the AR glasses DP output ready? (call after connecting) – a 3840x1200 display
// Note: NOT ready does NOT prove "not AR glasses" (the display may simply be unplugged/off);
//       the UVC signal is still needed to tell them apart.
bool carina_a1088_is_ar_display_ready(void);
```

5.14 Data-cap Glasses: Dual RGB Capture (device_type 19)

The data-cap glasses carry **two RGB channels** in addition to the four mono cameras. They are **not part of SLAM**: capture runs continuously for live preview, and pressing "start capture" additionally records MP4 files (one per channel).

```

// Latest frame of a given RGB channel as JPEG (for display)
// ch: 0 or 1      buf: caller-provided buffer      cap: buffer size in bytes
// returns bytes written (> 0); no frame / disabled / bad args / buffer too small → -1
int carina_al088_get_rgb_frame_jpeg(int ch, unsigned char* buf, int cap);

// — On-demand preview switch (this is what the UI's "Open RGB preview" button calls) —
// The capture channel serves BOTH preview and MP4 recording, so this only registers
// a demand: preview || recording → channel stays open; only when both are false it closes.
// Opening happens on a background thread (V4L2 open + negotiation takes ~100+ ms).
// While recording, passing 0 stops the display but keeps the channel (recording wins).
int carina_al088_set_rgb_preview(int on);          // 0 = registered; -1 = unsupported platform
int carina_al088_get_rgb_preview_state(void);      // 0=off 1=opening 2=open

// — Per-channel live frame rate (driver delivery rate, ~1 s rolling window) —
double carina_al088_get_rgb_fps(int ch);          // <0 = channel closed/unsupported
long long carina_al088_get_rgb_frame_count(int ch); // cumulative frames, <0 as above

```

Item	Detail
When capture starts	On demand: <code>carina_al088_set_rgb_preview(1)</code> (the UI button) or starting dataset capture; <code>rgb_auto_start: true</code> restores the old always-on behaviour, <code>save_rgb: false</code> is the master switch
Channels	<code>ch = 0 / 1</code> → <code>/dev/video22</code> / <code>/dev/video31</code> (fixed nodes on RK3588)
MP4 recording	Follows dataset saving: starts with <code>carina_al088_start_save_dataset(dir)</code> , finishes on <code>stop_save_dataset()</code> ; files are <code><dir>/rgb0.mp4</code> , <code><dir>/rgb1.mp4</code>
Encoding	Rockchip MPP hardware H.265 on RK3588, remuxed into MP4 without transcoding
Platform	Linux aarch64 (RK3588) only ; the capability is not built elsewhere and the API always returns -1
Dependencies	FFmpeg (MP4 muxing) + Rockchip MPP (hardware encoding)

Preview vs. recording (both share the capture channel; the SDK tracks them as demands):

Scenario	Capture channel	MP4
Preview only	opened by <code>set_rgb_preview(1)</code>	no file written
<code>start_save_dataset()</code>	opened automatically (even if preview is off)	writes <code><dir>/rgb0.mp4</code> , <code>rgb1.mp4</code>
"Close preview" while recording	stays open (recording wins)	keeps recording until <code>stop_save_dataset()</code>
Recording stopped and preview off	channel really closes	finalised (moov written, file playable)

Per-channel frame rate: `carina_al088_get_rgb_fps(0/1)` reports `rgb0` / `rgb1` separately (driver delivery rate, independent of preview). The `rgb0.fps` / `rgb1.fps` rows in the Pipeline table and the top status bar show these. Note `uvc.fps` is the UVC camera (data-cap glasses have none) — do not confuse the two.

Deployment notes (troubleshooting): RGB capture is isolated in a separate shared library `libcarina_rgb.so`; `libcarina_vio.so` itself does **not** depend on FFmpeg / MPP. Hence "does this platform support RGB capture" is expressed by ****whether the library can be dlopened****: on a non-RK3588 board (or with missing deps) the `dlopen` fails → silent degradation (one log line), the main program is unaffected. When shipping, make sure:

- libcarina_rgb.so sits next to libcarina_vio.so (or set LD_LIBRARY_PATH);
- FFmpeg runtime libs are available (libavformat / libavcodec / libavutil / libswscale / libswresample — shipped in the SDK lib/aarch64/);
- the board BSP provides librockchip_mpp.so.1 (MPP ships with the BSP, not with our packages).

5.15 Other Useful APIs

```
// — Dataset saving state —
int carina_a1088_is_saving_dataset();          // 1 = saving, 0 = not saving
int carina_a1088_is_config_driven_save();      // whether the current save was started by
                                              // config save_data=true (dynamic start returns 0)

// — Camera / trigger configuration —
bool carina_a1088_set_trigger_fps(uint8_t set_mode, int i_fps); // IMU hardware trigger fps
bool carina_a1088_set_slam_cam_en(uint8_t u_cam_mode);          // camera mask used by SLAM

// — Calibration state (call after connecting) —
bool carina_a1088_is_device_calibrated(); // calibration state cached in the session
// Note: the counterpart carina_a1088_is_calibrated(&cfg) briefly toggles USB to re-read it;
// do not call the two concurrently in one session.
```

The calibration workflow (enter calibration mode / AprilGrid capture / BA intrinsics+extrinsics / IMU-camera calibration / write back to device) lives in the separate header `api/carina_a1088_calibration.h` (`carina_a1088_calib_*`). That header is ****not shipped in the SDK package today**** (it is not in `CARINA_HEADERS` in `CMakeLists.txt`) — confirm with the team before offering on-site calibration to customers.

6. Configuration

6.1 Key Parameters

Key	Type	Description	Typical
<code>device_type</code>	int	Device type	10 / 20 / 30
<code>max_cameras</code>	int	Max camera count	2 ~ 4
<code>Camera.fps</code>	int	Camera FPS	30
<code>Imu.fps</code>	int	IMU rate	1000
<code>chip_tracking</code>	bool	On-chip tracking	true
<code>orb_database_path</code>	string	Vocabulary path	<code>./database.bin</code>

6.2 Templates

Config templates are in `config/templates/`:

- `a1088_base_template.yaml` — single-chip base config
- `four_cam_base_template.yaml` — quad-camera setup

6.3 Data-cap Glasses RGB (device_type 19)

Written into `custom_config.yaml` (defaults apply when absent):

Key	Type	Default	Description
save_rgb	bool	true	Enable dual RGB capture; false disables it entirely (no preview either)
rgb_auto_start	bool	false	Open the capture channel automatically at device start. false (default) = on demand: opened by the "Open RGB preview" button or when dataset capture starts
rgb_devices	string	/dev/video22,/dev/video31	Two V4L2 nodes, comma separated
rgb_codec	string	auto	auto / h265 / h264 / nv12; auto picks H.265 hardware encoding on RK3588
rgb_width	int	2320	Negotiated width
rgb_height	int	1744	Negotiated height
rgb_skip_frames	int	10	Frames dropped at startup (skip AE/AWB settling)

Example:

```
# Data-cap glasses (device_type=19), dual RGB
save_rgb: true
rgb_auto_start: false  # on-demand by default; set true to open at device start
rgb_devices: "/dev/video22,/dev/video31"
rgb_codec: auto
rgb_width: 2320
rgb_height: 1744
rgb_skip_frames: 10
```

7. Examples

7.1 Basic Usage

```
#include "carina_al088.h"
#include <stdio.h>
#include <unistd.h>

void on_pose(float *pose, double ts, void *user) {
    float *TWB = pose;
    printf("Pos: [%.3f, %.3f, %.3f]\n", TWB[12], TWB[13], TWB[14]);
}

void on_imu(float *imu, double ts, void *user) {
    // imu[0-2] = acc, imu[3-5] = gyro
}

void on_image(char *img0, char *img1, char *img2, char *img3,
              double ts, int w, int h, void *user) {
    // grayscale images, valid only during this callback
}

int main() {
    const char *config = "device_type: 10\nCamera.fps: 30\nImu.fps: 1000\n";

    int ret = carina_al088_init((char*)config, "./database.bin", -1, -1, -1);
    if (ret) return ret;

    ret = carina_al088_start(on_pose, NULL, on_imu, on_image, NULL, NULL, NULL);
    if (ret) { carina_al088_release(); return ret; }

    sleep(5); // run for 5 seconds

    carina_al088_stop();
    carina_al088_release();
    return 0;
}
```

7.2 Dual-Chip Device

```
// Just change the init function – everything else is identical
carina_double_al088_init((char*)config, "./database.bin", -1,-1,-1, -1,-1,-1);
```

7.3 Pause / Resume

Useful for focus switching, scene loading, etc.:

```
// Pause – preserve internal state, reduce CPU usage
carina_al088_pause();

// Do something expensive (e.g. load scene)
load_scene();

// Resume
carina_al088_resume();
```

7.4 Predicted Pose

```
#include <time.h>

float pose[16];
struct timespec ts;
clock_gettime(CLOCK_MONOTONIC, &ts);
double predict_time = ts.tv_sec + ts.tv_nsec * 1e-9;

// pose is a 4x4 matrix (column-major)
// [0,4,8,12] = rotation columns + translation
carina_al088_get_imu_pose(pose, predict_time + 0.0055);
```

7.6 Data-cap Glasses: Identification + RGB Preview (Linux aarch64)

```
#include "carina_al088.h"
#include <stdio.h>
#include <string.h>
#include <vector>

int main() {
    // 1) Identify the product BEFORE opening the device (no init needed)
    printf("probe: %s\n", carina_al088_probe_preopen_json());
    // {"al088_count":1,"uvc":false,"dp_display":false,"datacap_cams":true,...}

    int count = 0;
    const CarinaProductDesc* tbl = carina_product_table(&count);
    for (int i = 0; i < count; ++i)
        printf(" id=%d %s (type=%d)\n", tbl[i].id, tbl[i].name_short, tbl[i].device_type);

    // 2) Normal init / start (data-cap glasses are device_type 19)
    const char* config = "device_type: 19\nsave_rgb: true\n";
    if (carina_al088_init((char*)config, "./database.bin", -1, -1, -1)) return -1;

    // 3) After connecting, verify the selected product
    int32_t detected = -1, resolved = -1;
    int ok = carina_al088_check_product_match(5 /* data-cap glasses */, &detected, &resolved);
    printf("product match: %d (detected type=%d, resolved=%d)\n", ok, detected, resolved);

    if (carina_al088_start(NULL, NULL, NULL, NULL, NULL, NULL, NULL)) return -1;

    // 3.5) Open RGB preview on demand (same as the UI's "Open RGB preview" button)
    carina_al088_set_rgb_preview(1);
    // Opening is asynchronous: 0=off, 1=opening, 2=open
    while (carina_al088_get_rgb_preview_state() == 1) sleep(1);

    // 4) RGB preview: grab the latest JPEG frame (ch = 0/1)
    std::vector<unsigned char> buf(2 * 1024 * 1024);
    int n = carina_al088_get_rgb_frame_jpeg(0, buf.data(), (int)buf.size());
    if (n > 0) printf("rgb0 frame: %d bytes (JPEG)\n", n);
    else      printf("rgb0 no frame (preview not opened / not RK3588)\n");
    // buf now holds a plain JPEG byte stream – feed it to your UI or network stack
    // Per-channel refresh rate (driver delivery rate):
    printf("rgb0 %.1f fps / rgb1 %.1f fps\n", carina_al088_get_rgb_fps(0),
        carina_al088_get_rgb_fps(1));

    // 5) To record MP4: RGB follows dataset saving automatically
    carina_al088_start_save_dataset("/tmp/dataset"); // → /tmp/dataset/rgb0.mp4, rgb1.mp4
    sleep(5);
    carina_al088_stop_save_dataset();

    carina_al088_stop();
    carina_al088_release();
}
```

```
    return 0;
}
```

On non-RK3588 platforms or when capture is disabled, `carina_a1088_get_rgb_frame_jpeg()` always returns -1 — it never blocks and never fails loudly, so the caller can simply treat it as "no RGB capability".

8. Thread Safety & Data Lifetime

8.1 Thread Safety

- All API functions are **thread-safe**
- Callbacks run on internal worker threads — protect shared data accordingly

8.2 Data Lifetime Quick Reference

Function	Return Type	Lifetime	Free Required
<code>get_sn</code> , <code>get_cam_param</code> , <code>get_config</code>	<code>char*</code>	Static buffer, overwritten on next call	No
<code>get_sdk_version</code> , <code>get_firmware_version</code>	<code>char*</code>	Same as above	No
<code>read_custom_data</code>	<code>char*</code>	Same as above	No
<code>read_cus_para</code>	<code>uint32_t</code>	Same as above	No
<code>get_device_type</code>	<code>int32_t</code>	Value	No
<code>get_cur_status</code> , <code>detect_worn</code>	<code>bool</code>	Value	No
<code>get_display_level</code>	<code>int</code>	Value	No
<code>get_proximity_val</code> , <code>get_key_status</code>	<code>uint16_t</code>	Value	No
Callback data (pose, imu, image...)	Pointer	Valid during callback only	No
UVC callback data	<code>const uint8_t*</code>	Valid during callback only	No
<code>get_product_table_json</code> , <code>probe_preopen_json</code>	<code>char*</code>	Internally owned, read-only	No
<code>get_rgb_frame_jpeg</code>	<code>int</code>	Value (JPEG written into caller buffer)	No
<code>set_rgb_preview</code> , <code>get_rgb_preview_state</code> , <code>get_rgb_fps</code> , <code>get_rgb_frame_count</code>	value types	Value	No

8.3 Callback Data Rules

```
// ✅ CORRECT: copy data
void on_pose(float *pose, double ts, void *user) {
    static float last_pose[32];
    memcpy(last_pose, pose, sizeof(last_pose));
}

// ❌ WRONG: storing pointer (invalid after callback returns)
void on_pose(float *pose, double ts, void *user) {
    static float *stored = pose; // DANGEROUS!
}
```

9. Calibration C API (`api/carina_a1088_calibration.h`)

Full device calibration workflow: **enter calibration mode → AprilGrid capture → Ceres BA intrinsics+extrinsics → IMU-camera calibration → write back to device**. All algorithms live in the SDK (`libcarina_vio`); the desktop console's calibration panel calls exactly these functions.

⚠️ This header is **not shipped in the SDK package** (it is not in `CARINA_HEADERS` in `CMakeLists.txt`; this policy was confirmed on 2026-09-18). The symbols are exported from `libcarina_vio` on Linux / Windows / macOS; the Android build does not compile the calibration sources, so they do not exist there.

If a customer needs on-site calibration, provide the header through the internal process.

9.1 Callback types

```
typedef void (*CalibImageCallback)(const char* img0, const char* img1,
                                   const char* img2, const char* img3,
                                   int w, int h, double ts, void* user);

typedef void (*CalibImuCallback)(const float* acc, const float* gyro,
                                 double ts, void* user);

typedef void (*CalibDisplayCallback)(const char* img0, ...); // live preview during capture
typedef void (*CalibProgressCallback)(int percent, void* user); // progress 0..100
```

9.2 Call flow

```
// 1) Init (after the device is connected); pair with deinit
int carina_al088_calib_init(const char* config_file, const char* vocab_file);

// 2) Set before entering calibration mode
void carina_al088_calib_set_num_cameras(int n);          // 2 = A1088 stereo, 4 = A1088_4VGA
void carina_al088_calib_set_product_index(int idx);     // extrinsic init: 0=FT62 1=FT63 2=FT66 3=FT68
...

// 3) AprilGrid capture (only frames with a valid detection count)
int carina_al088_calib_start_capture(const char* config_json);
void carina_al088_calib_stop_capture(void);
int carina_al088_calib_get_capture_progress(int* captured, int* target, double* coverage);
int carina_al088_calib_set_display_callback(CalibDisplayCallback cb, void* user);

// 4) Calibration computation
int carina_al088_calib_run_camera_calib(CalibProgressCallback cb, void* user); // joint
intrinsic+extrinsic BA
int carina_al088_calib_run_imu_camera(double static_dur_sec, double motion_dur_sec,
                                      CalibProgressCallback cb, void* user); // hand-eye + joint
optimisation
int carina_al088_calib_get_camera_rms(int cam_id /*0..3*/, double* rms);
const char* carina_al088_calib_get_result_yaml(void); // result YAML (valid until the next
calibration call)

// 5) IMU static segments and manual parameter injection (~2 s static per face)
int carina_al088_calib_start_static_capture(void);
int carina_al088_calib_stop_static_capture(void);
int carina_al088_calib_get_static_mean(void);
int carina_al088_calib_get_current_imu(void);          // current IMU as a JSON-ish string (for live
display)
void carina_al088_calib_set_imu_result(double bg_x, double bg_y, double bg_z,
                                       double ba_x, double ba_y, double ba_z,
                                       double sx, double sy, double sz); // fix bg / ba / scale

// 6) Write back and clean up
int carina_al088_calib_write_to_device_data(const char* data, int len); // raw YAML straight to the
device
void carina_al088_calib_exit_mode(void); // stop capture / release resources / restore normal mode
void carina_al088_calib_deinit(void);
const char* carina_al088_calib_get_last_error(void);

// Calibration state query (briefly toggles USB to re-read the device config;
// do not run it concurrently with an active calibration session)
int carina_al088_is_calibrated(char** config_out);
```

10. carina_vio C++ API (feeding external image / IMU)

Use this when you **do not use the built-in drivers**: your own pipeline captures frames and IMU samples and feeds them into Carina (custom acquisition chains, dataset playback, embedding into your own framework). C++ API (`std::string` / `std::vector` / `std::function`); for pure C use `carina_a1088.h`.

```
int carina_vio_init(const std::string& config, const std::string& vocab_file_path);
int carina_vio_release();

// — Data input (device timestamps in seconds) —
void carina_vio_feed_images2(const char* img_left, const char* img_right, double timestamp);
void carina_vio_feed_images4(const char* img0, const char* img1,
                             const char* img2, const char* img3, double timestamp);
void carina_vio_feed_imu(const std::vector<float>& imu /*[ax,ay,az,gx,gy,gz]*/, double timestamp);

// — Pose output (predicttime = the moment to predict for, typically now + ~5 ms) —
int carina_vio_get_imu_pose(float* pose /*16*/, double predicttime);
int carina_vio_get_gl_pose(float* pose /*32*/, double predicttime);
int carina_vio_get_eyes_pose(float* pose_left /*16*/, float* pose_right /*16*/, double predicttime);

// — State / trajectory —
// Each trajectory point is 7 floats: [tx,ty,tz,qx,qy,qz,qw]
int carina_vio_get_trajectory(std::vector<float>& trajectory);
int carina_vio_get_slam_states_string(std::string& state); // YAML, can be persisted
int carina_vio_set_slam_states_string(const std::string& state);
int carina_vio_get_system_stage(); // 0=init 1=first mapping 2=loading map 3=relocalisation
4=continue mapping 5=re-mapping 6=saving map 7=shutdown -1=error
int carina_vio_get_saved_path(std::string& path);
int carina_vio_run_debug_state(); // 0=normal 1=terminate 2=pause
int carina_vio_reset(int level); // 0 = reset map only

// — Optional: register a pose callback instead of polling get_*_pose —
void carina_vio_set_pose_callback(std::function<void(float*, double)> callback); // (32 floats, ts)

// — Localisation mode (when a map already exists) —
int carina_vio_activate_localization_mode();
int carina_vio_deactivate_localization_mode();

// — Blocking event loop (typically used for dataset playback; call from the main thread) —
void carina_vio_run_win();
```

11. carina.h (primary C++ API overview)

`api/carina.h` is an earlier C++ layer (device mode + dataset playback). Use it **instead of** `carina_a1088.h`, not together: it registers all callbacks (pose / map / IMU / image) in one `carina_start()` call.

```

int carina_init(const std::string& config, const std::string& custom_config,
               const std::string& vocab_file_path,
               int vender_id = 0x04b4, int product_id = 0x00f1,
               int file_descriptor = -1, int bus_num = -1, int dev_addr = -1);
int carina_init_2088(...);           // A2088 variant
int carina_init2(...);              // explicit VID/PID variant
int carina_init_dataset(...);       // dataset playback mode
int carina_start(std::function<...> pose_cb, map_cb, vsync_cb, imu_cb, img_cb,
                 std::function<...> imu_pose_cb = nullptr);
int carina_pause();
int carina_stop();
int carina_release();

```

For day-to-day device integration prefer the `carina_a1088_*` C API in § 5 (pure C, identical across platforms, most complete documentation). This family is mainly used by internal demos/tools and dataset playback.

12. Other API families (index)

Families that are **not part of the A1088 device API** but are shipped in the SDK as well. Semantics are documented in each header's own comments.

Header	Family	Notes
<code>api/carina_a2088.h</code>	<code>carina_a2088_init</code> and others (3)	A2088 device
<code>api/carina_with_handle.h</code>	<code>carina_glasses_*</code> / <code>carina_handle_*</code> / <code>carina_led_ctl_*</code> (14)	handles / glasses (incl. LED control and handle IMU injection)
<code>api/carina_client.h</code>	<code>carina_client_*</code> (7)	TCP client (talks to a remote SLAM service)
rest of <code>api/carina.h</code>	<code>carina_get_config</code> / <code>get_sn</code> / <code>get_cam_param</code> / <code>get_render_para</code> / <code>get_config_des</code> / <code>set_application_state</code> / <code>set_rerun</code> / <code>set_vst_sharememory</code> / <code>set_vstframe_callback</code> / <code>update_custom_config</code> / <code>resume</code> / <code>compute_distortion</code> / <code>check_loop_state</code> ...	see § 11 and the header comments
<code>api/rgb_capture_abi.h</code>	<code>carina_rgb_*</code> (14)	Internal: the C ABI of <code>libcarina_rgb.so</code> , dlopen'ed by <code>carina_vio</code> ; not a customer API (use the § 5.14 <code>carina_a1088_*</code> wrappers instead)
<code>api/carina_a1088_calibration.h</code>	<code>carina_a1088_calib_*</code> (20)	calibration workflow, see § 9 (not shipped in the SDK , confirmed)
<code>api/carina_vio.h</code>	<code>carina_vio_*</code> (19)	feed external image/IMU, see § 10

