



FIRMWARE EXTERNAL INTERFACE
SPECIFICATION – HOST SIDE

gene core ファームウェア 外部インターフェース 仕様書

本書は、gene + Solist-AI 開発キットに含まれる、gene core に出荷時に書き込み済みのファームウェアが USB シリアル (テキストプロトコル) 経由でホスト PC へ提供する外部インターフェース (コマンド API) を規定します。





— 文書情報 / Document Information

対象製品 / Product	gene core (ESP32-S3)。gene + Solist-AI 開発キットの構成
対象 FW / Firmware	gene firmware for Solist-AI v0.1.273 (proto 1.0)
ホスト I/F / Interface	USB Type-C シリアル / UART 921600 bps (8N1)
対象ホスト OS	Windows / Linux
配布形態 / Distribution	バイナリ書込済みハードウェアとして提供（ソース非公開）。後に precompile ライブラリとユーザ拡張可能コードを提供予定。
版 / Version	Rev. 1.2



本書の位置づけ / Scope

本書は gene core（マスター / ホスト側）の物理・論理・機能と、PC から制御するための UART コマンド API を規定します。gene core が SPI 経由で制御する Solist-AI (ML63Q2537) スレーブ側の内部仕様は本書の対象外で、別資料で提供されます。

— 改訂履歴 / Revision History

版 / Rev.	日付 / Date	変更内容 / Changes
1.2	2026-10	対象ファームウェアを v0.1.273 に更新。配信中の <code>SET</code> 拒否と「停止 → 設定 → 再開」手順、 <code>START_STREAM</code> 応答によるフレーム構成の宣言、 <code>imu.mode</code> の追加と <code>imu.rate</code> の列挙化、IMU データロス時の自動停止 (<code>!IMU_ERR</code>)、配信中の状態表示 (<code>state=STREAMING</code> / LED 青)、録音の既定保存先 (内蔵フラッシュ) と MicroSD 不在時の自動切替、 <code>MIC_REC_START</code> の応答タイミング、 <code>FILE_DELETE</code> / <code>STORAGE_INFO</code> 、エラーコードを追加・修正。§ 6.9 を EEPROM (診断用、読み出しのみ) に変更。コマンドリファレンスを整理。



版 / Rev.	日付 / Date	変更内容 / Changes
1.1	2026-07	製品位置づけの記述を更新（p-ban.com 提供の PoC 向けハードウェアとして整理）。§ 6.7 マイク録音の <code>mic.storage</code> / ultrasonic 運用注記を明確化。コマンドリファレンスを整理。
1.0	2026-07	初版発行。ファームウェア v0.1.0 (Phase 6) 実装に基づき、物理仕様・論理構成・機能・UART コマンド・設定キーを規定。



商標 / Trademarks

Solist-AI™ は ROHM Co., Ltd. の商標です。gene は p-ban.com が提供する PoC 向けプラットフォームの名称です。その他、本書記載の会社名・製品名は各社の商標または登録商標です。



– 目次 / Contents

1	製品概要	7
1.1	gene core とは	8
1.2	開発キットの構成	9
1.3	出荷時の状態 (自動認識)	10
1.4	主な特長と想定用途	10
2	物理仕様	12
2.1	本体・外部インターフェース	13
2.2	開発キット付属のセンサー 4 種	13
2.3	ピン割り当て	14
3	システム構成 (論理)	16
3.1	役割分担と自動認識	17
3.2	起動時の自動認識フロー	17
3.3	通信モデルとデータフロー	18
3.4	動作状態・BUSY・設定永続化	19
4	機能仕様	22



4.1	機能一覧	23
4.2	センサーデータ取得 (ストリーミング)	23
4.3	マイク録音とファイル取得 (MIC)	26
4.4	AI・FFT (Solist-AI 連携)	28
4.5	機能と装着センサーの対応	29
5	はじめに (Get Started)	30
5.1	準備するもの	31
5.2	接続とシリアル設定	31
5.3	ターミナル / Python で試す	32
5.4	困ったときは	34
6	シリアルコマンド リファレンス	36
6.1	共通仕様・エラーコード	37
6.2	システム	40
6.3	設定 (SET / GET)	41
6.4	ストリーミング	43
6.5	マイク録音 (MIC)	45
6.6	ファイル・ストレージ (FILE / STORAGE)	45



6.7	AI 学習・推論 (ML)	48
6.8	信号処理 (FFT)	52
6.9	EEPROM (診断用)	54
7	設定キー リファレンス	55
7.1	IMU (BMI088)	56
7.2	ToF (VL53L5CX)	57
7.3	TH (SHT40-AD1B-R3)	57
7.4	マイク (SPH0641LU4H-1)	58
7.5	ストリーム共通 / 反映タイミング	59
8	付録	60
8.1	出荷時設定 (デフォルト値)	61
8.2	トラブルシューティング	61
8.3	用語集・仕様サマリ・参照リンク	63

1

CHAPTER 01

製品概要

gene core の位置づけ、開発キットの構成、出荷時の自動認識、主な特長と想定用途を示します。まず本章で全体像を把握してください。

1.1 gene core とは

1.2 開発キットの構成

1.3 出荷時の状態 (自動認識)

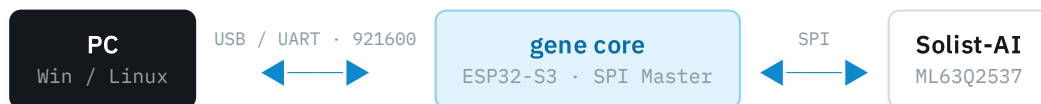
1.4 主な特長と想定用途



1 製品概要

1.1 gene core とは

gene core は、p-ban.com が提供する PoC (概念実証) 向けハードウェアです。gene + Solist-AI 開発キットの構成品の一つで、Solist-AI 連携ファームウェアを書き込んだ状態で提供します。



AI アクセラレータ Solist-AI モジュール (ML63Q2537) と ホスト PC を仲介する役割を持ちます。交換式のセンサー子基板から取得したデータを、AI アクセラレータ Solist-AI にリアルタイムで渡し、学習・推論・信号処理をオンデバイスで実行できます。ホスト PC とは USB シリアル (テキストプロトコル) で接続し、コマンドを送るだけで制御できます。



すぐ使える状態で出荷

ファームウェアは書き込み済み、センサーは起動時に自動認識される状態でお届けします。お客様によるビルドや書き込みは不要です。



1.2 開発キットの構成品

gene core は gene + Solist-AI 開発キットの構成品の一つです。開発キットには以下が含まれます。

#	品目	役割	備考
1	gene core 本体 (ESP32-S3)	メイン MCU / マスター制御	ファームウェア書込済み。 本書の対象
2	Solist-AI モジュール (ROHM ML63Q2537 搭載)	AI アクセラレータ (SPI スレーブ)	gene core と SPI 接続
3	IMU 子基板 (BMI088)	6 軸 加速度 + 角速度	自動認識対応
4	ToF 子基板 (VL53L5CX)	8×8 測距アレイ (ジェスチャ)	自動認識対応
5	MIC 子基板 (SPH0641LU4H-1)	デジタル MEMS マイク (PDM)	自動認識対応
6	Temp 子基板 (SHT40-AD1B-R3)	温度・相対湿度	自動認識対応
7	Conn ボード	SWD デバッグ・ 任意センサ接続 (J12 汎用コネクタ)	センサー子基板の代わりに 装着



センサーは排他装着

本体のセンサーコネクタには一度に 1 枚を取り付けて使用します。用途に応じて子基板を差し替えてください。



録音の保存先は 2 通り

MIC センサーで録音した音声は WAV として保存し、後からシリアル経由でダウンロードします。保存先の既定は本体内蔵フラッシュで、MicroSD を挿さなくても録音できます (16 kHz 音声で約 125 秒)。長時間録音や超音波録音 (75 kHz) では MicroSD を使用してください (→ [§ 4.3](#))。



1.3 出荷時の状態 (自動認識)

各センサー子基板は、コネクタに装着するだけで種別が自動判別されるよう出荷時に構成済みです。

- 起動時、gene core 本体が装着子基板の種別 (IMU / ToF / MIC / TH=Temperature & Humidity) を自動判別します。
- 判別結果に応じて該当ドライバが立ち上がり、共有ピンの役割も自動設定されます。お客様による種別設定は不要です。
- 詳細な自動認識フローは [§ 3.2](#) を参照してください。

1.4 主な特長と想定用途

AI

オンデバイス AI – 学習・推論を Solist-AI で実行。PC は司令塔に専念。

4-in-1

交換式マルチセンサー – 動作・空間ジェスチャ・音・環境を 1 つの本体で。

ASCII

テキストベース制御 – 改行区切りの ASCII コマンド。任意の言語・端末から扱える。

NVS

設定の永続化 – パラメータは本体内の不揮発メモリに保存され電源断後も保持。

PoC

想定用途 – 動作 / ジェスチャ認識、近接検知、異音検知、環境ロギング等の検証。

– 本書の範囲と別資料

範囲	扱い
gene core (マスター) の物理・論理・機能・制御 API	本書で扱う
各センサーの利用方法・データ形式	本書で扱う
Solist-AI (ML63Q2537, SPI スレーブ) 側の内部仕様	別資料で提供
ジェスチャ判定・音響解析などの上位アルゴリズム	お客様アプリ側 (本書対象外)



範囲	扱い
gene core ハードウェアの公開仕様	p-ban.com 公式情報を参照 (→ § 8)

2

CHAPTER 02

物理仕様

gene core を構成するハードウェアの物理的な仕様をまとめます。ピン配置はすべて公開可能な情報で構成しています。

2.1 本体・外部インターフェース

2.2 開発キット付属のセンサー 4 種

2.3 ピン割り当て



2 物理仕様

2.1 本体・外部インターフェース

出典: p-ban.com gene 製品仕様 (公開情報)。

項目	仕様
MCU	ESP32-S3-WR00M-1-N16R8 (Xtensa デュアルコア LX7, 最大 240 MHz)
SRAM / PSRAM	512 KB / 8 MB
フラッシュ	最大 16 MB (Quad SPI)
無線	Wi-Fi 802.11 b/g/n, BLE 5 (本 FW では未使用)
バッテリー	3.7 V / 400 mAh
USB	USB Type-C (USB 2.0 Full Speed) – PC 接続 + 充電
ストレージ	内蔵フラッシュ (録音の既定) + MicroSD スロット × 1 (マイク録音 WAV。長時間 / 超音波録音用)
センサーコネクタ	12 ピン (センサー子基板 1 枚を装着)
表示 / 操作	充電 LED × 1, リセットボタン × 1 (電源兼用)
サイズ / 動作電圧	40 × 40 × 19 mm / 3.0-3.6 V
マウント	アクションカムマウント対応

2.2 開発キット付属のセンサー 4 種

センサーは 排他装着です。コネクタには 1 枚のみ取り付けます。各子基板は装着するだけで種別が自動認識されます。



略称	部品 / メーカー	I/F	I ² C アドレス (7-bit) / 機能
IMU	BMI088 / Bosch	I ² C	acc 0x18 / gyro 0x68 - 6 軸
ToF	VL53L5CX / STMicroelectronics	I ² C	0x29 - 8x8 (or 4x4) 測距
MIC	SPH0641LU4H-1 / Knowles	PDM	(なし) - MEMS マイク
TH	SHT40-AD1B-R3 / Sensirion	I ² C	0x44 - 温度・相対湿度



IMU ボードの I²C 制御元切り替え

IMU 子基板のみ、ボード上の solder bridge (ジャンパ) をはんだ付けで切り替えることで、gene core ではなく Solist-AI モジュールから I²C 制御する構成に変更できます (既定は gene core 制御)。他のセンサーには本オプションはありません。

2.3 ピン割り当て

I²C (SCL/SDA) は全センサー共通で起動時に即時利用できます。一方、共有スロットピン (IO1 / IO2 / IO3) は装着子基板によって役割が変わります。本体は種別を判定するまでこれらのピンを設定しません (電氣的衝突を防ぐため)。

共通バス (全機種共通) と共有スロットピン

GPIO	ToF	IMU	MIC	TH
GPIO8 / 9	I2C_SCL / I2C_SDA - センサー共通, 1 MHz			
GPIO1	I2C_RST (出力)	—	PDM CLK (出力)	—
GPIO2	LPn (出力)	—	PDM DATA (入力)	—
GPIO3	INT (入力)	—	—	—
GPIO46	—	ボタン入力	—	—



本体内蔵デバイス (MicroSD / SPI / UART)

GPIO	信号	用途 / 備考
GPIO10-13	SD_CS / MOSI / CLK / MISO	MicroSD (SDSPI / SPI3) — <code>mic.storage=sd</code> での録音時のみ
GPIO21	SPI_CS	→ Solist-AI
GPIO35	SPI_MISO	← Solist-AI
GPIO36	SPI_SCLK	→ Solist-AI — Mode 0
GPIO37	SPI_MOSI	→ Solist-AI
GPIO38	SPI_READY	← Solist-AI (READY 信号)
GPIO43 / 44	UART0_TXD / RXD	PC ⇔ gene — 921600 bps

**gene ⇔ Solist-AI 配線 (参考)**

本体と Solist-AI は基板内で結線済みで、お客様の配線は不要です。SPI は 8-bit / MSB first / Mode 0 (CPOL=Low) で動作します。GPIO46 (IMU ボタン) は Solist-AI 側 (P83) にも分岐接続され、同じ信号を両側で取得できます。Solist-AI 側の詳細仕様は別資料を参照してください。

3

CHAPTER 03

システム構成 (論理)

各要素の役割、起動時の自動認識フロー、データの流れ、動作状態を説明します。物理仕様（第 2 章）と機能仕様（第 4 章）をつなぐ概念モデルです。

3.1 役割分担と自動認識

3.2 起動時の自動認識フロー

3.3 通信モデルとデータフロー

3.4 動作状態・BUSY・設定永続化



3 システム構成 (論理)

3.1 役割分担と自動認識

要素	役割
PC (Win / Linux)	司令塔。シリアルでコマンドを送り、応答・イベントを受け取る。上位アルゴリズム (判定・解析) はここ。
gene core (ESP32-S3)	マスター。PC コマンドの解釈、センサー駆動、Solist-AI への SPI 制御、データ整形・配信。
Solist-AI (ML63Q2537)	AI コプロセッサ (SPI スレーブ)。学習・推論・FFT を実行。詳細は別資料。
センサー子基板 (1 枚)	データ源。IMU / ToF / MIC / TH のいずれか。装着するだけで自己識別。

3.2 起動時の自動認識フロー

gene core は 起動時に 1 回、装着センサー子基板を自動判別します。

- 1 I²C バス初期化**
SCL/SDA のみ立ち上げ (共有ピン IO1/IO2/IO3 はまだ触らない)。
- 2 種別判定**
共通 I²C バス経由で装着子基板の種別 (IMU / ToF / MIC / TH) を判別する。
- 3 ドライバ起動**
種別に応じて IMU / ToF / MIC / TH ドライバを初期化。共有ピンの役割も種別に合わせて設定。
- 4 運用開始**
センサー共通タスクが、装着センサー種別に分岐して動作。



差し替え後は再起動または **SENSOR_RESCAN**

種別判定は起動時に行われます。子基板を差し替えたら本体を再起動するか、

>SENSOR_RESCAN で再検出してください (再起動不要)。未対応 / 未書込のセンサーは「デバイス 0 台」として扱われ、ストリーミングはできません。種別が変わると、前センサー固有の設定キー (例: **tof.***) は NVS に残りますが新センサーでは無視され、新センサーの設定が既定値 + 保存値で再適用されます。

3.3 通信モデルとデータフロー

PC と gene core はテキストベースのシリアルプロトコルで通信します。すべて改行 (**\n**, LF) 終端、1 行 = 1 メッセージ。通信は 921600 bps, 8N1, フロー制御なし。

種別	先頭	例	方向
コマンド	>	>START_STREAM 0	PC → gene
成功応答	<OK	<OK START_STREAM 0 mode=both rate=200 fields=...	gene → PC
エラー応答	<ERR	<ERR START_STREAM BUSY ...	gene → PC
非同期イベント	!	!SENSOR 0 12345 8 ...	gene → PC (自発)
ログ / コメント	#	# [12345] INFO: ...	gene → PC (無視可)

コマンドには 同期応答 (**<OK** / **<ERR**) が必ず返ります。センサーデータはストリーミング中に 非同期イベント (**!SENSOR**) として届きます。データは大きく 3 つの経路に分かれます。

経路	流れ	制御コマンド
A	センサー → gene → PC (リアルタイム配信)	START_STREAM / STOP_STREAM
B	PC → gene → (SPI) → Solist-AI → gene → PC (AI 演算)	ML_* / FFT_*

経路	流れ	制御コマンド
C	MIC → gene → 内蔵フラッシュ (/int) または MicroSD (/sdcard) (録音)、後で PC が取得	MIC_REC_* / FILE_DOWNLOAD / FILE_DELETE

3.4 動作状態・BUSY・設定永続化

gene core は内部に動作状態を持ち、`STATUS` コマンドで確認できます。



状態	意味	現行
INIT	初期化中	✓
IDLE	待機中 (コマンド受付可)	✓
STREAMING	センサーデータ配信中 (導出、変数には持たない)	✓
TRAINING	ML 学習中	予約
PREDICTING	ML 推論中	予約
ERROR	エラー状態 (致命的エラー時、任意の状態から遷移しうる)	✓

firmware が変数として保持する状態は INIT / IDLE / ERROR の 3 つです。STREAMING は変数に持たず、センサーが実際に配信しているかどうかから導出されます。`STATUS` の `state=` と RGB LED は同じ導出結果を見るので、両者が食い違うことはありません。導出にしているのは、配信が `STOP_STREAM` を経ずに終わる場合があるためです (IMU のデータロス停止など)。TRAINING / PREDICTING は将来拡張用に予約されており、`ML_TRAIN` / `ML_PREDICT` を実行しても `state=` は変わりません。



BUSY (排他制御)

一部の操作は同時実行できず、競合時は `<ERR ... BUSY ...>` を返します。照会系コマンド (`STATUS` / `VERSION` / `PING` / `GET` / `FILE_LIST` / `STORAGE_INFO`) はストリーミング中でも常時利用できます。`SET` は照会系ではなく、ストリーミング中は拒否されます (下記)。

状態	入る契機	影響
Streaming	START_STREAM 0 成功	再度の START_STREAM は BUSY。SET は全キーが BUSY。FILE_DOWNLOAD / FILE_DELETE / SENSOR_RESCAN も BUSY
Recording	MIC_REC_START 成功	再度の MIC_REC_START は BUSY。録音中ファイルへの FILE_DOWNLOAD は BUSY
ML busy	ML_TRAIN / ML_PREDICT	完了は ML_GET_BUSY で確認



設定の永続化 (NVS)

`SET` で変更した設定値は本体内の不揮発メモリ (NVS) に保存され、電源を切っても保持されます。次回起動時と同じ設定で立ち上がります (→ §7)。

ストリーミング中の SET は拒否されます

配信中に `>SET` を送ると、キーの種類にかかわらず `<ERR SET BUSY Cannot change configuration while streaming>` が返ります。設定を変えるときは次の手順で行います。

```

serial · 設定変更

>STOP_STREAM      ← 1. 止める
>SET imu.rate 400  ← 2. 変える
>START_STREAM 0    ← 3. 再開する

```



配信の途中で設定が変わると、センサーを再起動して数サンプルを落とすか、送出済みフレームの意味が黙って変わるかのどちらかになり、受信側は変更がどのフレームから効いたのかを判別できません。そのため、開始時に `>START_STREAM` の応答で宣言した構成は、そのストリームの間ずっと有効です。 `>GET` は配信中でも利用できます。

4

CHAPTER 04

機能仕様

gene core でできること – センサーデータ取得、AI 演算、信号処理、録音・ファイル取得 – を機能単位でまとめます。コマンドの厳密な書式は第 6 章を参照してください。

4.1 機能一覧

4.2 センサーデータ取得 (ストリーミング)

4.3 マイク録音とファイル取得 (MIC)

4.4 AI・FFT (Solist-AI 連携)

4.5 機能と装着センサーの対応

4 機能仕様

4.1 機能一覧

機能	概要	主なコマンド
センサーストリーミング	IMU / ToF / TH をリアルタイム配信	START_STREAM / STOP_STREAM
マイク録音	音声を WAV で本体内蔵フラッシュ（既定） / MicroSD に録音	MIC_REC_START / STOP
ファイル取得・削除	録音ファイル等をシリアルでダウンロード / 削除	FILE_LIST / FILE_DOWNLOAD / FILE_DELETE
ストレージ照会	保存先の空き容量と残り録音秒数	STORAGE_INFO
AI 学習・推論	Solist-AI で逐次学習 / 推論	ML_*
信号処理 (FFT)	Solist-AI で FFT 演算	FFT_*
センサー再検出	差し替え後に再判定 (再起動不要)	SENSOR_RESCAN
設定 / システム	パラメータ・状態・版数・接続確認	SET / GET / STATUS / VERSION / PING / RESET

4.2 センサーデータ取得 (ストリーミング)

>START_STREAM 0 で配信を開始すると、装着センサーに応じた !SENSOR イベントが連続的に届きます。>STOP_STREAM で停止します。共通形式は !SENSOR <device_id> <ts_ms> <ペイロード...> (device_id は常に 0)。

配信開始時にフレーム構成が宣言されます。>START_STREAM の応答は装着センサーに応じて、1 行に何がどの順で並ぶかを返します。配信中は >SET が拒否されるため、宣言された構成はそのストリームの間ずっと有効です。アプリ側はこの行を見てパーサを構成してください。



```

serial · START_STREAM 応答

IMU : <OK START_STREAM 0 mode=both rate=200 fields=ts_ms,ax,ay,az,gx,gy,gz
ToF : <OK START_STREAM 0 res=8 fields=ts_ms,res,d0..d63
TH : <OK START_STREAM 0 fields=ts_ms,t_milli_c,rh_milli_pct

```

ts_ms: gene 起動からの経過ミリ秒 (整数)。センサー内蔵 FIFO から読み出したフレームの推定サンプル時刻で、gene が行を送出した時刻ではありません。ミリ秒分解能のため、1000 Hz を超えるレートでは同じ **ts_ms** のフレームが並びます (異常ではありません。連続性はフレーム数とタイムスタンプのスパンで判定してください)。

種別	!SENSOR ペイロード	補足
IMU	ax ay az gx gy gz (mode=both) ax ay az (mode=accel) gx gy gz (mode=gyro)	16-bit raw。換算は imu.range_g / range_dps で PC 側。 imu.mode でフィールド数 が変わる
ToF	res d0 d1 ... dN	res=4/8、距離 mm、個数 N=res×res、欠測 0
TH	t_milli_c rh_milli_pct	1/1000 単位。例 27345→27.345 °C、 56123→56.123 %RH



数値はすべて整数

すべての数値は整数で送出されます。物理単位への換算は PC 側で行います (テキストプロトコルを整数のみに保つ設計)。MIC はストリーミング非対応で録音方式です (→ § 4.3)。

IMU 動作モードとレート

IMU は `imu.mode` で動作させるチップを選べます (`both` 既定 / `accel` / `gyro`)。

`mode=accel` と `mode=gyro` は wire 上どちらも 3 値で区別できないため、`>START_STREAM` 応答の `mode=` / `fields=` で判別してください。 `imu.rate` は列挙です (既定 200 Hz)。BMI088 に実在する ODR のみを受け付け、表に無い値は丸めずにエラーを返します。

<code>imu.mode</code>	選べる <code>imu.rate</code> (Hz)
both (既定)	100 / 200 / 400
accel	25 / 50 / 100 / 200 / 400 / 800 / 1600
gyro	100 / 200 / 400 / 1000 / 2000

`200` は 3 モードすべてで有効です。モードを切り替えるときは一度 `>SET imu.rate 200` を挟むと、組み合わせ不正を避けられます。

IMU データロス時の自動停止 (`!IMU_ERR`)

IMU はサンプルを 1 つでも落としたら配信を停止し、`!IMU_ERR <code> seq=<n>` で理由を通知します。落としたデータを黙って続けるより止めるほうが、学習・推論のデータセットが壊れないため、この設計にしています。

コード	意味
OVERRUN	送信が追いつかず、gene 内部の配信バッファが満杯になった
MISSED_TICK	FIFO の読み出しが次の読み出し周期までに終わらなかった
I2C_ERROR	センサーの I ² C 読み出しがリトライ後も失敗した
FIFO_OVERFLOW	センサー内蔵 FIFO がフレームを捨てた
PAIR_BACKLOG	<code>mode=both</code> で加速度と角速度のレート差が開き、ペアにできなくなった
NO_DATA	500 ms の間どちらのセンサーからもデータが現れない



- `seq=<n>` は配信開始からのサンプル通し番号です。
- このイベントを受け取った時点で配信は既に止まっています。以降 `!SENSOR` は届きません。再開するには `>START_STREAM 0` を送り直してください (`>STOP_STREAM` は `NOT_STREAMING` を返します)。
- 累積回数は `>STATUS` の device 行 (`overrun=` / `missed=`) で確認でき、`>START_STREAM` のたびにクリアされます。

4.3 マイク録音とファイル取得 (MIC)

MIC はリアルタイム配信ではなく「録音 → 保存 → ダウンロード」方式です (PDM の高データレートを UART で送り切れないため)。保存先は `mic.storage` で選択します。

設定	保存先	容量の目安	用途
internal (既定)	本体内蔵フラッシュ <code>/int</code>	4 MB ≒ 125 秒 (16 kHz)	MicroSD 不要。音声・ウェイクワード
sd	MicroSD <code>/sdcard</code>	カード容量による	長時間録音、超音波 (75 kHz)



MicroSD が無いときは内蔵へ自動的に切り替わります

`mic.storage=sd` を選んでいてもカードを認識できない場合、gene は内蔵フラッシュへ切り替えて録音を継続し、`mic.storage` の値自体を `internal` に書き換えます (NVS に保存されます)。MIC が使えなくなるより録れるほうが有用なためです。カードを挿したら、あらためて `>SET mic.storage sd` で選び直してください。このため、要求した保存先に録れていると仮定しないでください。実際の保存先は `<OK MIC_REC_START` の `path=` (先頭が `/int/` か `/sdcard/` か)、または `>STORAGE_INFO` の `backend=` が正となります。

典型フロー (既定の内蔵フラッシュの場合):

```

serial · MIC flow

>MIC_REC_START

<OK MIC_REC_START path=/int/mic_rec_12345_1.wav    ← 録音開始まで待ってから返る
... (録音中) ...

```



serial · MIC flow

>MIC_REC_STOP

<OK MIC_REC_STOP path=/int/mic_rec_12345_1.wav

!MIC_REC_DONE path=/int/mic_rec_12345_1.wav ← 完了通知（ポーリング不要）

>FILE_LIST /int

<OK FILE_LIST entry=mic_rec_12345_1.wav size=1500000

>FILE_DOWNLOAD /int/mic_rec_12345_1.wav

<OK FILE_DOWNLOAD size=1500000 crc32=0xa1b2c3d4

<... size バイトのバイナリ列（エスケープ無し） ...>

>FILE_DELETE /int/mic_rec_12345_1.wav ← CRC 照合後に削除して領域を空ける

<OK FILE_DELETE path=/int/mic_rec_12345_1.wav



- **<OK MIC_REC_START** は録音が実際に始まってから返ります。待機・ノイズ区間の破棄・WAV ヘッダ書き込みを終えた時点なので、応答まで最大 3 秒かかります (通常 1.5~2 秒)。他のコマンドと同じ短いタイムアウトを設定すると、正常な録音開始を失敗と誤判定するので注意してください。
- WAV のサンプルレートは `mic.clock_hz` から自動算出 (`clock_hz / 64`)。1024000 (既定) → 16 kHz (音声 / ウェイクワード)、4800000 → 75 kHz (超音波 / 機械異音検知。`mic.storage=sd` 推奨)。
- 録音は `mic.max_record_seconds` (既定 60 秒) と保存先の空き容量で自動停止。残り録音可能秒数は `>STORAGE_INFO` の `rec_seconds_left=` で取得できます。
- 録音中に電源が切れた場合、最後にストレージへ書き出した以降のデータは失われる可能性があります (`MIC_REC_STOP` / 自動停止での正常終了時に WAV が確定)。
- ダウンロードのバイナリは `<OK` 行の `size=` バイトをそのまま受信し、`crc32=` (IEEE 802.3) で検証します。
- CRC 照合が通ってから `FILE_DELETE` してください (転送失敗時のデータ喪失を防ぐため)。内蔵フラッシュは 4 MB しかないので、取得後の削除を習慣にすると容量切れによる録音の早期停止を避けられます。
- MIC 装着時に `>START_STREAM 0` を送ると `NOT_SUPPORTED` を返します (録音コマンドを使用)。

4.4 AI・FFT (Solist-AI 連携)

gene core は SPI マスターとして Solist-AI を駆動し、逐次学習 (online learning) と推論、FFT を実行します。Solist-AI 連携機能はセンサー子基板に依存しません。

```

serial . ML flow

>ML_INIT <inst> <input> <hidden> <output> ...      ← 全 11 引数は §6.7 参照
>ML_TRAIN <inst> input=[...] target=[...]
>ML_GET_BUSY                                     ← 0 になるまで待つ
>ML_PREDICT <inst> input=[...]
>ML_GET_RESULT <inst>                            ← 推論結果を取得

```



serial · ML flow

>ML_GET_LOSS <inst>

← 損失を取得

- 重み (β) と共分散行列 (P) の取得・設定 (`ML_GET/SET_WEIGHT_*`) に対応し、学習済みモデルの保存・復元が可能です。
- 浮動小数は SPI 転送時に bfloat16 に変換されます (高速・低帯域)。
- **FFT**: PCM サンプル列を Solist-AI に渡し FFT を実行 (`FFT_INIT` → `FFT_START` → `FFT_GET_RESULT` 、サイズ最大 1024)。

4.5 機能と装着センサーの対応

機能	IMU	ToF	MIC	TH
<code>START_STREAM</code> ストリーミング	✓	✓	× 録音	✓
<code>MIC_REC_*</code> 録音	—	—	✓	—
<code>FILE_*</code> / <code>STORAGE_INFO</code> / <code>ML_*</code> / <code>FFT_*</code> / <code>SENSOR_RESCAN</code> / 設定	✓ センサー非依存 (どのセンサーでも利用可)			

5

CHAPTER 05 ・ GET STARTED

はじめに (Get Started)

gene core を PC につないで最初のコマンドを送るまでを案内します。対象ホスト OS は Windows / Linux です。

5.1 準備するもの

5.2 接続とシリアル設定

5.3 ターミナル / Python で試す

5.4 困ったときは



5 はじめに (Get Started)

5.1 準備するもの

- gene core 本体（ファームウェア書込済み）と、使用するセンサー子基板 1 枚
- USB Type-C ケーブル（データ通信対応のもの）
- MIC の音声録音は本体に内蔵フラッシュで動作し MicroSD は不要。長時間録音 / 超音波録音には MicroSD カードを使用する（非同梱・お客様手配、`mic.storage=sd` で切替）
- PC (Windows / Linux) とシリアルターミナル、または Python (任意)

5.2 接続とシリアル設定

1 センサーを装着

使うセンサー子基板を本体コネクタに 1 枚取り付けます（差し替えた場合は再起動するか `>SENSOR_RESCAN` で再認識）。MIC で長時間 / 超音波録音をする場合は MicroSD も挿入（短い音声録音は本体に内蔵フラッシュに録れるので不要）。

2 USB Type-C で PC に接続し電源 ON

PC 上にシリアルポートが現れます（Linux: `/dev/ttyUSB0` / `/dev/ttyACM0`、Windows: `COM3` 等）。

3 STATUS で待機 (IDLE) を確認

シリアルポートを開き `>STATUS` を送り、`state=IDLE` が返れば待機完了です。ポートが現れない場合は USB-UART ブリッジ用ドライバ (CP210x / CH34x 等) を導入してください。

項目	値
ボーレート	921600（固定 - 不一致だと文字化け）
データ / パリティ / ストップ	8 / なし / 1 (8N1)
フロー制御	なし
改行コード	LF (\n) を送信



5.3 ターミナル / Python で試す

Linux では `screen` / `minicom` / `picocom` 等、Windows では Tera Term / PuTTY が使えます (送信改行を LF に設定)。接続後、以下を 1 行ずつ送信します。

```
terminal

# picocom の例 (改行を LF で送る)

picocom -b 921600 --omap crlf /dev/ttyUSB0

>PING 1
<OK PONG 1
>VERSION
<OK VERSION proto=1.0 fw=0.1.273 build="..." solist.input_max=256
>STATUS
<OK STATUS uptime=12345 state=IDLE solist=ok devices=1
<OK STATUS device:0 type=T0F bus=i2c addr=0x29 fw=VL53L5CX_2.0.1
```

スクリプトや自動評価には Python (pyserial) が便利です。< 行 (同期応答) ・ ! 行 (イベント) ・ # 行 (ログ) を先頭文字で振り分けて処理します。

```
python · pyserial

import serial

ser = serial.Serial("/dev/ttyUSB0", 921600, timeout=1) # Windows は "COM3" 等

def cmd(line):
    ser.write((line + "\n").encode())
    while True:
```



python · pyserial

```
resp = ser.readline().decode(errors="replace").strip()

if resp.startswith("<"): # 同期応答を待つ
    return resp

print(cmd(">PING 1"))      # -> <OK PONG 1
print(cmd(">STATUS"))      # -> <OK STATUS uptime=... state=IDLE ...
```

最初のストリーミングは `>START_STREAM 0` で開始、`>STOP_STREAM` で停止。`!SENSOR` の内容はセンサー種別で異なります (→ § 4.2)。

terminal · streaming

```
>START_STREAM 0
<OK START_STREAM 0 res=8 fields=ts_ms,res,d0..d63 ← フレーム構成の宣言 (ToF の例)
!SENSOR 0 12345 8 ...
>STOP_STREAM
<OK STOP_STREAM
```



- **<OK START_STREAM** の行がフレーム構成を宣言します。1 行に何がどの順で並ぶかは装着センサーと設定で変わるので、この行を見てパーサを構成してください (IMU は `mode=` `rate=` `fields=`、ToF は `res=` `fields=`、TH は `fields=`)。
- 配信中は `>STATUS` の `state=` が `STREAMING`、RGB LED は青点灯になります。停止すると `IDLE` / 緑に戻ります。
- 配信中の `>SET` は拒否されます (`<ERR SET BUSY ...`)。設定を変えるときは `>STOP_STREAM` → `>SET` → `>START_STREAM 0` の順で行ってください。
- IMU はデータを 1 つでも落とすと配信を自分で止めて `!IMU_ERR` を送ります。受け取ったら `!SENSOR` はもう届きません (→ § 4.2)。
- MIC の録音では、`<OK MIC_REC_START` が録音が実際に始まってから返ります (最大 3 秒)。`path=` が実際の保存先です。取得して CRC を照合したら `FILE_DELETE` で消しておく、次の録音が容量切れで早期停止するのを防げます (→ § 4.3)。

5.4 困ったときは

症状	確認
応答が文字化け	ボーレートが 921600 か。8N1 / フロー制御なしか。
何も返らない	ポート選択は正しいか。コマンドが <code>></code> 始まり・LF 終端か。
<code>devices=0</code>	センサーが装着・認識されているか。差し替え後は再起動か <code>>SENSOR_RESCAN</code> 。
<code>START_STREAM</code> が <code>NOT_SUPPORTED</code>	MIC 装着時。録音コマンドを使用。
<code>BUSY</code> が返る	既にストリーミング / 録音中。停止してから再実行。
<code>>SET</code> が <code>BUSY</code>	配信中は設定を変更できない。 <code>>STOP_STREAM</code> → <code>>SET</code> → <code>>START_STREAM 0</code> 。
<code>>SET imu.rate</code> が <code>INVALID_KEY</code>	<code>imu.rate</code> は列挙で、 <code>imu.mode</code> ごとに選べる値が違う (→ § 7.1)。丸めは行われない。
配信が勝手に止まる	<code>!IMU_ERR</code> が出ていないか確認。データロス検出時は配信を自動停止する (→ § 4.2)。



症状	確認
<code>MIC_REC_START</code> が返らない	録音開始まで最大 3 秒かかる。タイムアウトを延ばす。

詳しくは § 8.2 トラブルシューティングを参照してください。

6

CHAPTER 06 ・ COMMAND REFERENCE

シリアルコマンド リファレンス

gene core が受け付けるすべての UART コマンドの書式と応答をまとめます。本リファレンスはファームウェア v0.1.273 の実装に基づきます。

6.1 共通仕様・エラーコード

6.2 システム

6.3 設定 (SET / GET)

6.4 ストリーミング

6.5 マイク録音 (MIC)

6.6 ファイル・ストレージ (FILE / STORAGE)

6.7 AI 学習・推論 (ML)

6.8 信号処理 (FFT)

6.9 EEPROM (診断用)



6 シリアルコマンド リファレンス

6.1 共通仕様・エラーコード

項目	値
物理 / ボーレート	USB Type-C シリアル / 921600 bps
フレーミング	8N1 / フロー制御なし / 行終端 \n (LF)
コマンド	>COMMAND [引数...]
成功 / エラー応答	<OK COMMAND [...] / <ERR COMMAND <CODE> <msg>
非同期イベント / ログ	!EVENT [...] / # [ts] LEVEL: msg (無視可)

数値・配列の表記: 配列引数は `input=[v1,v2,v3,...]` (角括弧・カンマ区切り)、配列応答も `[v1,v2,...]` 形式。ML 系の浮動小数は SPI 転送時に bfloat16 (上位 16bit) へ変換されます。

プロトコル共通規約

- **文字コード:** コマンド・応答・イベント・ログ、およびファイル名はすべて ASCII 印字可能文字のみ。非 ASCII (日本語等) は用いない。
- **行終端:** 双方向とも 1 行 = 1 メッセージで `\n` (LF) 終端 (PC→gene・gene→PC のいずれも)。
- **<msg> フィールド:** `<ERR>` やログの `<msg>` は ASCII 印字可能文字のみで、LF・タブを含まない。最大長は 1 行 **256 バイト**。
- **命名の例外:** 応答の先頭語は原則コマンド名と一致するが、`PING` のみ応答が `<OK PONG ...` となる。
- **バイナリ転送の例外:** 「1 行 = 1 メッセージ」原則の例外として、バイナリを伴うコマンド (現状 `FILE_DOWNLOAD` のみ) は `<OK` 行の LF 直後から `size=` バイトの生バイナリが続き、その直後に次のメッセージが始まる。バイナリ区間にエスケープ・区切りは無い。
- **crc32 表記:** `crc32=0x<hex>` は常に 0 埋め 8 桁 (IEEE 802.3, poly 0xEDB88320)。



エラーコード一覧

コード	意味
INVALID_CMD	不明なコマンド
INVALID_PARAM	引数の不足 / 形式不正
INVALID_STATE	現在の状態では実行できない（前提コマンド未実行等）
INVALID_KEY	未知の設定キー。値が範囲外・形式不正のときもこれが返る（キーの誤りと値の誤りを区別しない）
NO_MATCH	glob (<code><prefix>.*</code> / <code>*</code>) で該当キーが 0 件
BUSY	使用中につき実行不可（配信中の <code>SET</code> / 二重の <code>START_STREAM</code> など）
NOT_INIT	対象インスタンスが未初期化
NOT_READY	サブシステム未初期化
NOT_FOUND	デバイス / ファイルが見つからない
NOT_IMPLEMENTED	コマンドは存在するがハンドラ未実装
NOT_SUPPORTED	この構成では非対応の操作（例: MIC 装着時の <code>START_STREAM</code> ）
NOT_STREAMING / NOT_RECORDING	対象が動作中でない
TIMEOUT	期限内に処理が完了しなかった
DEVICE_ERROR	SPI / I ² C / ハードウェア障害
IO_ERROR	ファイル / ストレージ障害
NO_MEM	メモリ不足
SIZE_EXCEEDED	データがバッファ上限超過
VERIFY_FAILED	書き込み後の照合不一致



コード	意味
LINE_TOO_LONG	1 行が行バッファ (64 KB) を超えた（その行は破棄されます）



6.2 システム

PING >PING <ts> System	
目的	接続確認。 <ts> はラウンドトリップ計測用の任意整数（エコーされる）
応答	<OK PONG 123

VERSION >VERSION System	
目的	プロトコル / ファームウェア版数の取得 + Solist-AI capability (<code>input_max</code>) の取得
応答	<OK VERSION proto=1.0 fw=<major>.<minor>.<build> build="<日付>" solist.input_max=<uint unknown>
補足	<code>solist.input_max</code> : Solist-AI が受け付ける <code>>ML_INIT</code> の <code>input_size</code> 上限 (bf16 element 数)。Solist-AI 未接続 / capability 取得失敗時は文字列 <code>unknown</code> 。

STATUS >STATUS System	
目的	システム状態 + 接続センサー。1 行目がサマリ、以降が接続デバイス（1 台ごと）
応答	<OK STATUS uptime=<ms> state=<STATE> solist=ok devices=<N> <OK STATUS device:<id> type=<TYPE> bus=<bus> [addr=<addr>] fw=<ver> [overrun=<n> missed=<n>]
補足	<code>state</code> =INIT/IDLE/STREAMING/TRAINING/PREDICTING/ERROR（配信中は <code>STREAMING</code> 。TRAINING / PREDICTING は予約で、現行は遷移しません）、 <code>type</code> =TOF/IMU/MIC/TH、 <code>bus</code> =i2c (IMU/ToF/TH) / pdm (MIC)。 <code>addr=</code> は I ² C センサーのみ。 <code>overrun=</code> / <code>missed=</code> は IMU のみで、配信中に取りこぼした回数の累計（ <code>>START_STREAM</code> のたびに 0 に戻る、→ § 6.4 <code>!IMU_ERR</code> ）。例 (IMU): <code>device:0 type=IMU bus=i2c addr=0x18 fw=<ver> overrun=0 missed=0</code> 。 <code>devices=0</code> のときは 1 行目のみ。 <code>imu.mode</code> / <code>imu.rate</code> などの設定値は <code>STATUS</code> には出ません。 <code>>GET</code> で取得してください。

RESET >RESET <soft hard> System	
目的	<code>soft</code> = 状態を INIT に戻す / <code>hard</code> = 本体再起動
エラー	未知の reset type INVALID_PARAM

RESCAN >SENSOR_RESCAN System	
目的	装着センサーを再検出する（再起動不要）。子基板を差し替えた後に使用。現在のセンサーを解放し、EEPROM を読み直して新しい種別のドライバを起動する
応答	<OK SENSOR_RESCAN kind=<KIND> detected=<KIND> eeprom=<0 1> [device:0 type=... bus=... [addr=...] fw=...]
補足	<code>kind</code> =再検出後に有効になったセンサー（IMU/TOF/MIC/TH/NONE）、 <code>detected</code> =EEPROM が返した種別（ <code>kind ≠ detected</code> はドライバ初期化失敗）、 <code>eeprom</code> =EEPROM が応答したか（ <code>0</code> = 見つからない）
エラー	配信中 / 録音中 BUSY（先に STOP_STREAM / MIC_REC_STOP） 応答なし TIMEOUT

6.3 設定 (SET / GET)

SET >SET <key> <value> NVS 永続化	
目的	設定変更（NVS に永続化）。キー一覧は § 7 参照。例: <code>>SET imu.rate 100</code> → <code><OK SET imu.rate 100</code>
エラー	配信中（キーの種類を問わず） BUSY 未知のキー / 値が範囲外・形式不正 INVALID_KEY キー / 値の欠落 INVALID_PARAM
補足	配信中は <code><ERR SET BUSY Cannot change configuration while streaming</code> 。 <code>>STOP_STREAM</code> → <code>>SET</code> → <code>>START_STREAM 0</code> の順で行ってください（→ § 3.4）。 <code>>GET</code> は配信中でも利用できます。



GET

>GET <key> / >GET * / >GET <prefix>.*

System

目的	設定取得。単一キー、全キー ()、プレフィックス glob ( など) に対応
応答	<OK GET imu.rate 100 (単一キーは key value 形式) <OK GET tof.resolution=8 tof.freq=15 ... (glob / 全取得は key=val を空白区切り)
エラー	glob に一致なし NO_MATCH 単一キーで未知のキー INVALID_KEY

6.4 ストリーミング

START >START_STREAM <device_id> Stream	
目的	センサーデータ配信を開始する。配信中は !SENSOR イベントが連続で届く。応答がフレーム構成を宣言する
応答	<OK START_STREAM 0 mode=both rate=200 fields=ts_ms,ax,ay,az,gx,gy,gz (IMU) <OK START_STREAM 0 res=8 fields=ts_ms,res,d0..d63 (ToF) <OK START_STREAM 0 fields=ts_ms,t_milli_c,rh_milli_pct (TH)
引数	<device_id> : 常に 0 (将来の複数センサー同時装着に備えた予約)
イベント	!SENSOR <id> <ts> <payload>
補足	fields= が !SENSOR 1 行の並び。装着センサーと設定で変わるので、アプリはこの行を見てパーサを構成する。IMU は mode= (both/accel/gyro) と rate= も宣言する (accel と gyro はどちらも 3 値で wire 上は区別できないため、判別はこの行で行う)。配信中は >SET が拒否されるため、宣言した構成はそのストリームの間ずっと有効。配信中は state=STREAMING 、RGB LED が青。 overrun= / missed= のカウンタはこのコマンドでクリアされる。 <ts_ms> はセンサー内蔵 FIFO から読み出したフレームの推定サンプル時刻で、1000 Hz を超えるレートでは同じ値の行が並ぶ (異常ではない)
エラー	引数不正 / デバイス無し INVALID_PARAM / NOT_FOUND 既に配信中 BUSY MIC 装着時 NOT_SUPPORTED 期限内に配信が始まらない TIMEOUT
注記	NOT_FOUND は device_id が違う、センサーが未検出、あるいはドライバが計測開始を拒否した場合。 TIMEOUT はドライバが何も問題を報告しないまま、期限内に配信が始まらなかった場合。前者は装着や配線を、後者はデバイスの応答性を疑う



STOP >STOP_STREAM Stream	
目的	配信停止
イベント	なし（配信停止。以降 <code>!SENSOR</code> は届かない）
エラー	配信していない NOT_STREAMING
	期限内に停止しない TIMEOUT
注記	<code>TIMEOUT</code> は停止要求を受け付けたが、センサータスクが期限内に配信ループを抜けなかった場合。配信は続いている可能性があるので、 >STATUS の <code>state=</code> で確認する（ STREAMING なら継続中）

EVENT !IMU_ERR <code> seq=<n> Stream	
目的	IMU のみ。サンプルを 1 つでも落とすと配信を停止し、理由を通知する。 <code> = <code>OVERRUN / MISSED_TICK / I2C_ERROR / FIFO_OVERFLOW / PAIR_BACKLOG / NO_DATA</code> （意味は § 4.2）。 seq=<n> は配信開始からのサンプル通し番号
補足	受信した時点で配信は既に停止しており、以降 <code>!SENSOR</code> は届かない。再開は >START_STREAM 0 の再発行（ >STOP_STREAM は <code>NOT_STREAMING</code> ）。累計は >STATUS の device 行（ <code>overrun=</code> / <code>missed=</code> ）で読める



6.5 マイク録音 (MIC)

REC >MIC_REC_START [basename] MIC							
目的	録音開始。 <code>basename</code> 省略時は自動命名 → <code><OK MIC_REC_START path=<フルパス></code>						
補足	<code><OK</code> は録音が実際に始まってから返る。待機・ノイズ区間の破棄・WAV ヘッダ書き込みを終えた時点で送出されるので、「準備完了」ではなく「録音中」を意味する。応答まで通常 1.5～2 秒、最大 3 秒かかるため、他のコマンドと同じ短いタイムアウトでは正常な開始を失敗と誤判定する。WAV の先頭は <code><OK</code> を送出した時点で揃う（送出中に拾った音は破棄）。<code>path=</code> は実際の保存先。<code>mic.storage=sd</code> でも MicroSD を認識できなければ内蔵フラッシュへ自動的に切り替わるため、<code>/int/</code> が返ることがある。この <code>path=</code> の先頭を正として扱う（→ §7.4）						
エラー	<table> <tr> <td>MIC 未装着</td><td>NOT_FOUND</td></tr> <tr> <td>録音中</td><td>BUSY</td></tr> <tr> <td>保存先を mount できない / ファイルを開けない / 録音開始タイムアウト</td><td>IO_ERROR</td></tr> </table>	MIC 未装着	NOT_FOUND	録音中	BUSY	保存先を mount できない / ファイルを開けない / 録音開始タイムアウト	IO_ERROR
MIC 未装着	NOT_FOUND						
録音中	BUSY						
保存先を mount できない / ファイルを開けない / 録音開始タイムアウト	IO_ERROR						
注記	<code>IO_ERROR</code> が返った場合でも内蔵フラッシュへの切り替えは済んでいることがある。 <code>>GET mic.storage</code> で現在値を確認する						

REC >MIC_REC_STOP MIC					
目的	録音停止（WAV 確定まで待機）。完了時に <code>!MIC_REC_DONE path=<フルパス></code> を発火				
エラー	<table> <tr> <td>録音していない</td><td>NOT_RECORDING</td></tr> <tr> <td>ストレージ障害</td><td>IO_ERROR</td></tr> </table>	録音していない	NOT_RECORDING	ストレージ障害	IO_ERROR
録音していない	NOT_RECORDING				
ストレージ障害	IO_ERROR				

6.6 ファイル・ストレージ (FILE / STORAGE)

録音データの保存先は 2 つあり、パスで指定します。



保存先	パス	備考
本体内蔵フラッシュ	/int	<code>mic.storage=internal</code> (既定)。4 MB
MicroSD	/sdcard	<code>mic.storage=sd</code>

FILE >FILE_LIST <path>		File
目的	ディレクトリ内のファイル一覧（通常ファイルのみ）を取得する	
応答	<OK FILE_LIST count=<N> <OK FILE_LIST entry=<name> size=<bytes> ... (entry 行を N 行、1 行 1 エントリ)	
補足	<code>count</code> 行の entry が 1 行 1 ファイルで返る。 <code>name</code> は空白を含まない ASCII。ホストは <code>count</code> と受信した entry 行数の一致で完了を判定できる	
エラー	path 不足 / 不在 / IO INVALID_PARAM / NOT_FOUND / IO_ERROR	

FILE >FILE_DOWNLOAD <path>		File
目的	ファイルをバイナリ転送。<OK FILE_DOWNLOAD size=<bytes> crc32=0x<8hex> の後に size バイトの生バイナリが続く	
補足	<code>crc32</code> =IEEE 802.3 (poly 0xEDB88320)。バイナリにはエスケープ・区切りが無く、 <code>size=</code> が正となります。	
エラー	配信 / 録音中ファイル BUSY 不在 / IO / メモリ NOT_FOUND / IO_ERROR / NO_MEM	



FILE >FILE_DELETE <path>

File

目的	ファイルを削除する。録音 → ダウンロード → 削除の運用（特に容量の小さい内蔵フラッシュ）を想定。例: <code>>FILE_DELETE /int/foo.wav</code>
応答	<code><OK FILE_DELETE path=<フルパス></code>
補足	<code>FILE_DOWNLOAD</code> の CRC 照合が通ってから削除する（転送失敗時のデータ喪失を防ぐため）
エラー	path 欠落 不在 通常ファイルでない / 削除失敗 配信中 / 録音中のファイル INVALID_PARAM NOT_FOUND IO_ERROR BUSY

STORAGE >STORAGE_INFO

File

目的	現在の録音保存先の容量と、残り録音可能秒数を返す。引数なし
応答	<code><OK STORAGE_INFO backend=<sd internal> total=<bytes> free=<bytes> usable=<bytes> sample_rate=<hz> rec_seconds_left=<sec></code>
補足	<code>backend</code> =実際の保存先（internal / sd）、 <code>total</code> / <code>free</code> =総容量 / 空き容量（バイト）、 <code>usable</code> = <code>free</code> - 64 KB（録音の安全マージンを引いた値）、 <code>sample_rate</code> =現在の PCM サンプルレート（ <code>mic.clock_hz / 64</code> ）、 <code>rec_seconds_left</code> =残り録音可能秒数（ <code>usable ÷ (sample_rate × 2)</code> ）。 <code>backend=</code> が要求した保存先と一致すると仮定しないこと。 <code>mic.storage=sd</code> でも MicroSD を認識できなければ内蔵へ切り替わる。この応答の <code>backend=</code> が正。 例: <code>backend=internal total=4194304 free=4130816 usable=4064768 sample_rate=16000 rec_seconds_left=127</code>
エラー	内蔵フラッシュも mount できない / 配信中に MicroSD が使えない NOT_FOUND



6.7 AI 学習・推論 (ML) Solist-AI 連携

Solist-AI 上のニューラルネットに対する逐次学習・推論の中核コマンド群です。1 台あたり複数のネットワーク（`<inst>` = インスタンス番号）を保持でき、各コマンドで対象を指定します。共通エラー：

`NOT_READY`（未初期化） / `DEVICE_ERROR`（SPI 障害） / `SIZE_EXCEEDED`（配列が転送上限超過）。



INIT

>ML_INIT <inst> <in> <hidden> <out> <act> <loss> <forget> <seed>
<scale_a> <scale_g> <leak>

ML

目的

インスタンス `<inst>` のオンライン学習ネットワークを構築する。以降の学習・推論の前提となる。全 11 引数をスペース区切りで指定する

引数

<inst> インスタンス番号 (0 / 1。同時に扱える数は構成依存 → 備考)
<in> 入力次元数 (inputSize)
<hidden> 隠れ層ユニット数 (hiddenSize)
<out> 出力次元数 (outputSize)
<act> 活性化関数 (activationFunction): 0=Linear / 1=Sigmoid / 2=ReLU
<loss> 損失関数 (lossFunction): 0=MAE / 1=MSE
<forget> 学習率 / 忘却係数 (forgettingFactor, 実数 → bfloat16。典型値 0.01)
<seed> 重み初期化の乱数シード (seed, 整数)
<scale_a> 重み α スケール (scaleAlpha, 実数 → bfloat16)
<scale_g> 重み γ スケール (scaleGamma, 実数 → bfloat16。ESN モード用)
<leak> リークレート (leakRate, 実数 → bfloat16。ESN モード用)

補足

`scale_g` / `leak` は Echo State Network 系モデル設定用。通常の前向きネットワークとして使う場合は意味を持たず、既定値のままでよい。実数引数 (`forget` / `scale_a` / `scale_g` / `leak`) はファーム側で bfloat16 に変換して保持される

備考

`in` / `hidden` の上限は構成に依存する (構成 A: インスタンス 0 のみ・`in` ≤ 512・`hidden` ≤ 64 / 構成 B: インスタンス 0,1・`in` ≤ 256・`hidden` ≤ 64)。初期化に成功すると以降ファームが `in` / `hidden` / `out` を保持し、`ML_GET_RESULT`・サイズ取得の応答長はこれらから自動計算される

応答

<OK ML_INIT <inst>

エラー

引数不足 / 範囲外 (`in` / `hidden` 上限超過・0, `out`=0 等)
Solist-AI 通信失敗

INVALID_PARAM

DEVICE_ERROR



RESET >ML_RESET <inst> / >ML_CLEAR_RAM

ML

目的	ML_RESET = 指定インスタンスの重み・状態を初期化（構成は保持）。 ML_CLEAR_RAM = 全インスタンスを破棄し RAM を解放
応答	<OK ML_RESET <inst> / <OK ML_CLEAR_RAM
エラー	未初期化 NOT_READY 不正な inst INVALID_PARAM

TRAIN >ML_TRAIN <inst> input=[...] target=[...]

ML

目的	1 サンプルで逐次学習（1 ステップ）。input は入力ベクトル、target は教師ベクトルで、それぞれ ML_INIT の入力／出力ユニット数と一致させる。演算は非同期で、完了は ML_GET_BUSY が 0 を返すことで確認する
応答	<OK ML_TRAIN <inst>
エラー	未初期化 NOT_READY 長さ不一致 / 形式不正 INVALID_PARAM 配列が上限超過 SIZE_EXCEEDED 演算中 BUSY

PREDICT >ML_PREDICT <inst> input=[...]

ML

目的	入力ベクトルから推論を実行。演算は非同期で、完了は ML_GET_BUSY で確認し、結果は ML_GET_RESULT で取得する
応答	<OK ML_PREDICT <inst>
エラー	未初期化 NOT_READY 長さ不一致 INVALID_PARAM 演算中 BUSY



STATUS >ML_GET_BUSY

ML

目的	Solist-AI が学習／推論を実行中かを返す（1 = 演算中 / 0 = 完了）。ML_TRAIN / ML_PREDICT 後のポーリングに用いる
応答	<OK ML_GET_BUSY <flag>

RESULT >ML_GET_RESULT <inst>

ML

目的	直近の ML_PREDICT の出力ベクトルを取得。1 行目にサマリ、続けて出力データ行が届く
応答	<OK ML_GET_RESULT <inst> size=<n> <データ行>
エラー	結果未確定 / 演算中 未初期化 INVALID_STATE / BUSY NOT_READY

LOSS >ML_GET_LOSS <inst>

ML

目的	直近の学習ステップの損失値を取得。学習の収束確認に用いる
応答	<OK ML_GET_LOSS <inst> <loss>
エラー	未初期化 / 未学習 NOT_READY / INVALID_STATE

WEIGHT >ML_SET_WEIGHT_BETA <inst> input=[...] / >ML_GET_WEIGHT_BETA <inst>

ML

目的	重み係数 β の設定／取得。学習済みモデルの保存・復元（ホスト側でのシリアルライズ）に用いる。取得時は 1 行目のヘッダに続けてデータ行が届く
応答	<OK ML_SET_WEIGHT_BETA <inst> <OK ML_GET_WEIGHT_BETA <inst> size=<n> + データ行
エラー	未初期化 長さ不一致 配列が上限超過 NOT_READY INVALID_PARAM SIZE_EXCEEDED



WEIGHT

>ML_SET_WEIGHT_P <inst> input=[...] / >ML_GET_WEIGHT_P <inst>

ML

目的	共分散行列 P の設定／取得。β と対で保存・復元することで逐次学習の内部状態をまるごと引き継げる。取得時は 1 行目のヘッダに続けてデータ行が届く
応答	<OK ML_SET_WEIGHT_P <inst> <OK ML_GET_WEIGHT_P <inst> size=<n> + データ行
補足	P の総バイト数は <code>hidden × hidden × 2</code> 。 <code>ML_GET_WEIGHT_P_SIZE</code> で事前取得できる
エラー	未初期化 NOT_READY 長さ不一致 INVALID_PARAM 配列が上限超過 SIZE_EXCEEDED

WEIGHT

>ML_GET_WEIGHT_BETA_SIZE <inst> / >ML_GET_WEIGHT_P_SIZE <inst>

ML

目的	β / P 配列の総バイト数を取得。取得コマンドの前に受信バッファを確保する用途に使う (β = <code>hidden × out × 2</code> / P = <code>hidden × hidden × 2</code>)
応答	<OK ML_GET_WEIGHT_BETA_SIZE <inst> <size> <OK ML_GET_WEIGHT_P_SIZE <inst> <size>
エラー	未初期化 NOT_READY

6.8 信号処理 (FFT)

Solist-AI 連携

入出力とも int16 の実数系 FFT。点数（サイズ）は 1～1024、窓関数は 矩形（なし）または Hann。

`FFT_INIT` でサイズ・窓を設定し、`FFT_START` → 完了待ち → `FFT_GET_RESULT` の順で使う（同じ設定で繰り返す場合は再初期化不要）。演算アクセラレータは AI と共有のため、AI 演算と FFT 演算は同時に実行できない。



INIT >FFT_INIT <size> <window>

FFT

目的	FFT の点数と窓関数を設定する（1 回のみ。同一設定で繰り返す場合は再初期化不要）
引数	<size> FFT 点数（1～1024） <window> 窓関数: 0 =なし（矩形） / 1 =Hann
応答	<OK FFT_INIT
エラー	size=0 / >1024, window>1 INVALID_PARAM

START >FFT_START input=[...]

FFT

目的	int16 PCM を投入して FFT 演算を開始する。input の要素数は FFT_INIT の size と一致させる。演算は非同期で、完了は FFT_GET_BUSY で確認する
応答	<OK FFT_START
エラー	FFT_INIT 未実行 INVALID_STATE 長さ不一致 / int16 範囲外 INVALID_PARAM AI/FFT 演算中 BUSY

STATUS >FFT_GET_BUSY

FFT

目的	FFT 演算が実行中かを返す（1 =演算中 / 0 =完了）。AI と FFT は同一アクセラレータを共有するため、この busy 状態は ML_GET_BUSY と同一の値を返す
応答	<OK FFT_GET_BUSY <flag>

**RESULT** >FFT_GET_RESULT

FFT

目的	直近の FFT 結果（int16 配列）を取得する。1 行目にサイズ、続けてデータ行が届く
応答	<OK FFT_GET_RESULT size=<n> <データ行>
エラー	FFT_INIT 未実行 / 結果未確定 演算中

INVALID_STATE

BUSY

6.9 EEPROM (診断用)

センサー子基板の識別 EEPROM を読み出します。通常運用では使用不要です（出荷時に書き込み済みで、起動時に自動認識されます）。装着したセンサーが認識されない場合の切り分けや、個体識別に使用します。

**読み出しのみ**

EEPROM は出荷時に書き込み済みで、お客様による書き換えは想定していません。本製品で利用できるのは読み出しのみです。

EEPROM >EEPROM_READ

System

目的	識別 ID (1 バイト) を読む（診断用）
応答	<OK EEPROM_READ id=0x02
補足	id : 0x01 =IMU / 0x02 =ToF / 0x03 =MIC / 0x04 =TH
エラー	EEPROM が応答しない

NOT_FOUND

EEPROM >EEPROM_SERIAL

System

目的	工場プログラム済みの 16 バイトシリアル番号を読む（個体識別）
応答	<OK EEPROM_SERIAL sn=<32 桁 hex>

7

CHAPTER 07 ・ CONFIGURATION

設定キー リファレンス

`>SET <key> <value>` で変更、`>GET <key>` で取得します。値は NVS に保存され電源断後も保持。配信中 (`START_STREAM` 実行中) の `SET` は、キーの種類を問わず拒否されます。



7 設定キー リファレンス

未知のキー、および値が範囲外・形式不正のときは `<ERR SET INVALID_KEY <key>`、キー / 値の欠落は `<ERR SET INVALID_PARAM ...` を返します。 `>GET *` で全キーをまとめて取得できます。



配信中の SET は拒否されます

配信中は `<ERR SET BUSY Cannot change configuration while streaming` が返ります。設定を変えるときは `>STOP_STREAM` → `>SET` → `>START_STREAM 0` の順で行ってください（理由は § 7.5）。 `>GET` は配信中でも使えます。

7.1 IMU (BMI088)

キー	型	既定	取り得る値 / 説明
imu.mode	str	both	both / accel / gyro。動作させるチップ。 <code>imu.rate</code> の選択肢と <code>!SENSOR</code> のフィールド数が変わる
imu.rate	int	200	下表の列挙値のみ（mode 依存）。サンプリングレート
imu.range_g	int	6	3/6/12/24（加速度レンジ ±g）
imu.range_dps	int	500	125/250/500/1000/2000（角速度レンジ ±dps）

`imu.rate` は範囲ではなく列挙です。BMI088 に実在する ODR のみを受け付け、表に無い値（例 300）は丸めずに `<ERR SET INVALID_KEY imu.rate` を返します。

<code>imu.mode</code>	選べる <code>imu.rate</code> (Hz)	<code>!SENSOR</code> の値
both（既定）	100 / 200 / 400	6 値 (ax ay az gx gy gz)
accel	25 / 50 / 100 / 200 / 400 / 800 / 1600	3 値 (ax ay az)
gyro	100 / 200 / 400 / 1000 / 2000	3 値 (gx gy gz)



- `imu.mode` と `imu.rate` は互いを検証します。組み合わせが不正になる `SET` (例: `mode=both` のまま `imu.rate 1600`) は失敗します。
- `200` は 3 モードすべてで有効です。モードを切り替えるときは一度 `>SET imu.rate 200` を挟むと、順序によらず必ず成功します。
- `mode=accel` と `mode=gyro` は wire 上どちらも 3 値で区別できません。`>START_STREAM` 応答の `mode=` / `fields=` で判別してください。

7.2 ToF (VL53L5CX)

キー	型	既定	取り得る値 / 説明
<code>tof.resolution</code>	int	8	4 / 8 (グリッド解像度 4×4 / 8×8)
<code>tof.freq</code>	int	15	1–60 Hz (8×8 は実質 15 まで)
<code>tof.sharpener</code>	int	5	0–99 % (エッジ先鋭化)
<code>tof.target_order</code>	str	closest	closest / strongest
<code>tof.integration_ms</code>	int	5	1–1000 (autonomous モード時のみ有効。 continuous モードでは無視)
<code>tof.ranging_mode</code>	str	continuous	continuous / autonomous

7.3 TH (SHT40-AD1B-R3)

キー	型	既定	取り得る値 / 説明
<code>th.rate</code>	int	1	1–10 Hz (取得レート)
<code>th.repeatability</code>	str	high	low / medium / high (high $\approx$ 8.2 ms / low $\approx$ 1.6 ms)

7.4 マイク (SPH0641LU4H-1)

キー	型	既定	取り得る値 / 説明
mic.clock_hz	int	1024000	1024000-4800000。PCM レート = clock_hz / 64 (1024000→16 kHz 音声 / 4800000→75 kHz 超音波)。既定は voice (16 kHz)
mic.max_record_seconds	int	60	1-3600 (録音の自動停止までの最大秒数)
mic.storage	str	internal	<code>internal</code> = 本体内蔵フラッシュ (<code>/int</code> 、SD 不要・~125 s@16 kHz)、 <code>sd</code> = MicroSD (<code>/sdcard</code> 、長時間用)。残り録音可能時間は <code>>STORAGE_INFO</code> で取得



内蔵フラッシュ + 超音波 (4.8 MHz) は非推奨

150 KB/s の連続書き込みに内蔵フラッシュが追いつかずサンプルを取りこぼし、再生が速く・音質が劣化します。超音波録音では `>SET mic.storage sd` を使ってください。

MicroSD が使えないときの自動切り替え

`mic.storage=sd` を選んでいても MicroSD を認識できない場合、gene は内蔵フラッシュへ切り替えて録音を継続し、`mic.storage` の値自体を `internal` に書き換えます (NVS に保存され、次回起動時も `internal` です)。カードが無いことはストレージの問題であってセンサーの問題ではなく、内蔵フラッシュには必ず録音できるため、MIC が使えなくなるより録れるほうを選んでいきます。この切り替えが起きると、以下がすべて `internal` を指すように揃います。

確認方法	切り替え後の値
<code>>GET mic.storage</code>	<code>internal</code>
<code>>STORAGE_INFO</code> の <code>backend=</code>	<code>internal</code>
<code><OK MIC_REC_START</code> の <code>path=</code>	<code>/int/...</code>



要求した保存先に録れていると仮定しないでください。上記のいずれかで実際の保存先を確認してください。MicroSD を挿したら、あらためて `>SET mic.storage sd` で選び直します。

7.5 ストリーム共通 / 反映タイミング

キー群	配信中の SET	反映
imu.* / tof.* / th.* / stream.*	不可 (BUSY)	停止中に設定し、次の <code>>START_STREAM</code> から有効
mic.*	可（録音は配信ではないため）	次の録音開始時にスナップショット

なぜ配信中の SET を拒否するのか

配信の途中で設定が変わると、センサーを再起動して数サンプルを落とすか、送出済みのフレームの意味が黙って変わるかのどちらかになります。どちらの場合も、受信側は変更がどのフレームから効いたのかを判別できません。そのため gene は配信中の変更を受け付けず、`>STOP_STREAM` → `>SET` → `>START_STREAM 0`（応答が新しい構成を宣言する）という手順に統一しています。これにより、`>START_STREAM` の応答で宣言された構成は、そのストリームが終わるまで変わらないことが保証されます。

録音 (`MIC_REC_*`) は配信ではないため、`mic.*` は録音中でも `SET` できます。ただし反映は次の録音からです（録音中の設定は開始時にスナップショットされます）。

すべてのキーは NVS に保存され、再起動後も保持されます。

8

APPENDIX

付録

出荷時設定、トラブルシューティング、用語集、仕様サマリ、参照リンク、別資料と注意事項をまとめます。

8.1 出荷時設定 (デフォルト値)

8.2 トラブルシューティング

8.3 用語集・仕様サマリ・参照リンク



8 付録

8.1 出荷時設定 (デフォルト値)

項目	値
ファームウェア版数	0.1.273 (プロトコル 1.0)
ボーレート	921600 bps (8N1)
センサー識別	起動時に自動認識
RGB LED 自動表示	状態連動 (IDLE=緑 / STREAMING=青 / ERROR=赤点滅)
録音の保存先	本体内存フラッシュ (<code>mic.storage=internal</code> 、約 125 s@16 kHz)
設定値	§ 7 の「既定」列のとおり

8.2 トラブルシューティング

症状	原因の可能性	対処
文字化けする	ボーレート不一致	921600 / 8N1 / フロー制御なしを確認
応答が返らない	ポート / 改行 / プレフィックス	正しいポート選択、 <code>></code> 始まり・LF 終端を確認
<code>STATUS</code> が <code>devices=0</code>	センサー未認識	装着確認。差し替え後は再起動か <code>SENSOR_RESCAN</code>
差し替えが反映されない	再検出未実行	本体を再起動するか <code>SENSOR_RESCAN</code> を実行
<code>START_STREAM</code> → <code>NOT_SUPPORTED</code>	MIC 装着中	<code>MIC_REC_START</code> を使用
<code>BUSY</code> が返る	ストリーミング / 録音中	停止後に再実行

症状	原因の可能性	対処
録音できない / <code>IO_ERROR</code>	保存先が使用不可	<code>>STORAGE_INFO</code> で <code>backend=</code> と空き容量を確認。 <code>mic.storage=sd</code> で SD が無い場合は内蔵へ自動切替される
<code>FILE_DOWNLOAD</code> / <code>FILE_DELETE</code> が <code>BUSY</code>	配信中 or 録音中ファイル	停止後、または別ファイルで実行
<code>>SET</code> が <code>BUSY</code>	配信中は設定変更不可	<code>>STOP_STREAM</code> → <code>>SET</code> → <code>>START_STREAM 0</code>
<code>>SET imu.rate</code> が <code>INVALID_KEY</code>	<code>imu.rate</code> は列挙で、 <code>imu.mode</code> により選択肢が変わる (丸めは行われない)	§ 7.1 の表から選ぶ。 <code>200</code> は全モードで有効
配信が勝手に止まる	IMU のデータロス検出による自動停止	<code>!IMU_ERR</code> の <code><code></code> を確認。 <code>>START_STREAM 0</code> で再開 (§ 4.2)
<code>MIC_REC_START</code> の応答が遅い	仕様。録音開始まで待ってから <code><OK></code> を返す	タイムアウトを 3 秒以上にする
SD に録ったはずが <code>/int</code> に保存される	SD を認識できず内蔵へ自動切替	カード挿入を確認し <code>>SET mic.storage sd</code> 。実保存先は <code>path=</code> / <code>>STORAGE_INFO</code> の <code>backend=</code> で確認
録音がすぐ止まる	保存先の空き容量不足	<code>>STORAGE_INFO</code> の <code>rec_seconds_left=</code> を確認。 <code>FILE_DELETE</code> で不要ファイルを削除
ML 系が <code>NOT_READY</code>	初期化前	<code>ML_INIT</code> を先に実行
ML 系が <code>DEVICE_ERROR</code>	Solist-AI との SPI 障害	再起動。別資料 (Solist-AI 側) を確認



8.3 用語集・仕様サマリ・参照リンク

gene

p-ban.com が提供する PoC 向けプラットフォームの名称

gene core

本書の対象。ホストボード (ESP32-S3 搭載)。gene + Solist-AI 開発キットの構成品の一つ

Solist-AI [ML63Q2537](#)

ROHM の AI アクセラレータ搭載 MCU (SPI スレーブ)

ToF [Time-of-Flight](#)

光の往復時間で距離を測る方式 (VL53L5CX)

IMU

慣性計測装置。加速度 + 角速度 (BMI088)

TH [Temperature & Humidity](#)

温湿度センサー (SHT40-AD1B-R3)。ファームウェア上の種別識別子。ハードウェア仕様書では Temp 子基板

PDM

Pulse Density Modulation。デジタルマイクの出力方式

bfloat16

16bit 浮動小数フォーマット (float32 の上位 16bit)

NVS

ESP32 の不揮発設定ストレージ

内蔵フラッシュ [/int](#)

本体 MCU 内のフラッシュに確保した録音領域 (4 MB)。MicroSD 不要

ODR

Output Data Rate。センサーが出力するサンプリング周波数

ODL [On-Device Learning](#)

オンデバイス逐次学習

仕様サマリ

項目	値
メイン MCU	ESP32-S3 (Xtensa デュアルコア, 最大 240 MHz)
AI コプロセッサ	ROHM ML63Q2537 (Solist-AI, SPI スレーブ)
PC 接続	USB Type-C / UART 921600 bps (テキスト)



項目	値
センサー I/F	I ² C 1 MHz (IMU/ToF/TH) / PDM (MIC)
ストレージ / サイズ	内蔵フラッシュ (録音の既定、約 125 s@16 kHz) + MicroSD (SDSPI、長時間・超音波録音用) / 40 × 40 × 19 mm
対象 ホス ト OS	Windows / Linux



参照リンク・別資料

gene (p-ban.com) 製品仕様 / ROHM Solist-AI ブランド・ML63Q2537 製品ページ、および主要センサー (Bosch BMI088 / ST VL53L5CX / Knowles SPH0641LU4H-1 / Sensirion SHT40-AD1B-R3) のメーカー公開データシートを参照。Solist-AI (ML63Q2537) SPI スレーブ側の詳細仕様、および Solist-AI のフラッシュ (学習済み重み・設定の保存領域) は別資料で提供します。



— 免責事項 / Disclaimer

本書に記載された情報は、発行時点のものであり、予告なく変更されることがあります。本書の内容については正確を期していますが、**当社は本書の記載内容に関して明示・黙示を問わずいかなる保証も行わず**、本書の使用または使用不能から生じるいかなる損害（逸失利益、事業の中断、データの損失等を含む）についても責任を負いません。本製品は評価・開発（PoC）用途を目的としており、医療機器・航空宇宙・原子力など、人命に関わる用途や高い信頼性が要求される用途への使用を想定していません。

本製品の仕様、外観、付属品は改良のため予告なく変更される場合があります。お客様が本製品を組み込んだ最終製品の適合性・安全性については、お客様ご自身の責任において十分に評価・検証してください。本製品のファームウェアは非公開バイナリです。ソースコードの提供・改変は行いません。記載のピン配置・仕様は公開可能な情報の範囲です。

注記・特記事項 / Notes & Special Remarks

- ※1 本書中の数値（アドレス・サイズ・タイミング・コマンド書式等）は本インターフェース仕様を規定するものであり、ホスト側（PC アプリケーション）はこれらの記載に基づいて実装してください。本書に記載のない挙動には依存しないでください。
- ※2 Solist-AI の推論精度は学習データ・動作環境に依存し、特定の性能を保証するものではありません。
- ※3 ファームウェア更新により、将来コマンドの追加や機能拡張が行われる場合があります。本書はファームウェア v0.1.273 時点の内容です。
- ※4 輸出に際しては、関連する輸出関連法令を遵守してください。該当する場合は事前に許可が必要です。

著作権表示 / Copyright

本書の著作権は当社に帰属します。当社の事前の書面による許可なく、本書の全部または一部を複製・転載・改変・翻訳・再配布することを禁じます。



© 2026 P板.com (p-ban.com). All rights reserved.

Solist-AI™ is a trademark of ROHM Co., Ltd. その他、本書に記載の会社名・製品名は各社の商標または登録商標です。