AI-Kit Cloud API Integration Guide

ESP32 Edge Controller Telemetry Upload, Authentication, Retry Logic and Firmware Acceptance Testing

Item	Details
Document Number	AST-AIK-API-001
Document Title	AI-Kit Cloud API Integration Guide
Version	1.0
Status	Approved for Firmware Integration
Company	Altius Surya Tech Pvt Ltd
Project	AI-Kit Edge Controller Platform
Primary Firmware Target	ESP32 with RS485 Modbus and WiFi/GSM/4G connectivity
Cloud Endpoint	https://ai.altiussuryatech.in/api/telemetry/ingest
Related Documents	AST-AIK-FW-001 Telemetry Integration Specification v1.1; AST-AIK-FW-002 INVT Register Mapping v1.0
Date	01/07/26

Purpose: This guide gives firmware developers the practical steps, examples, acceptance checks, and troubleshooting guidance required to connect an ESP32-based AI-Kit Edge Controller to the AI-Kit Cloud Platform.

Document Control

Item	Details
Owner	K N Vishwanath / Altius Surya Tech Pvt Ltd
Audience	Firmware developer, embedded systems integrator, test engineer, field commissioning engineer
Security Classification	Controlled - Do not include actual API keys in source code, logs, screenshots, email threads, or shared repositories.
Credential Handling	Device-specific API keys are supplied separately through restricted provisioning records. This document contains no production API key.
Supersedes	New document
Reference Baseline	AI-Kit Edge Controller Telemetry Integration & Cloud Communication Specification v1.1

Revision History

Version	Date	Description	Status
1.0	01/07/26	Initial firmware developer guide for ESP32 cloud integration, including HTTPS telemetry upload, authentication, payload format, retry logic, buffering, examples, commissioning checks, and FAT checklist.	Approved for Firmware Integration

Table of Contents

1. Purpose and Scope

This document is the practical API integration guide for firmware developers implementing the AI-Kit Edge Controller cloud communication function. It converts the approved telemetry specification into direct implementation steps for ESP32 firmware development.

- Read telemetry from the VFD or field controller using RS485 Modbus RTU.
- Normalize the values using the selected VFD driver, initially VFD_TYPE = INVT.
- Build the AI-Kit telemetry JSON payload.
- Authenticate using Device ID and the assigned API key.
- Upload telemetry to AI-Kit Cloud using HTTPS POST.
- Retry failed uploads and buffer telemetry locally during network outages.
- Demonstrate successful dashboard updates and pass the acceptance checklist.

1.1 What This Document Is

- A firmware implementation guide for connecting ESP32 devices to the production AI-Kit Cloud.
- A controlled engineering reference for cloud API usage, retry logic, and acceptance testing.
- A companion document to the telemetry specification and the INVT register mapping document.

1.2 What This Document Is Not

- It is not a device credential sheet. Actual API keys must be supplied separately in restricted provisioning records.
- It is not a replacement for the INVT Modbus register mapping document.
- It does not approve remote VFD write control, remote start/stop, or cloud-to-device command execution.

2. Related Documents

Document No	Document Title	Purpose
AST-AIK-FW-001	AI-Kit Edge Controller Telemetry Integration & Cloud Communication Specification v1.1	Defines telemetry fields, cloud endpoint, authentication header, sampling interval, buffering requirement, VFD abstraction, and firmware deliverables.
AST-AIK-FW-002	AI-Kit Register Mapping - INVT v1.0	Defines INVT Modbus communication parameters, register block, status register, fault register, scaling, mapping, and reserved write registers.
AST-AIK-PRV-001	AI-Kit Controller Device Credentials	Restricted provisioning record containing device-specific Device IDs and API keys.

3. Implementation Summary for Firmware Developer

Step	Developer Action	Expected Result
1	Load DEVICE_ID, API_KEY, FIRMWARE_VERSION, VFD_TYPE, Modbus parameters and API endpoint from non-volatile configuration.	Firmware starts without hard-coded credentials.
2	Initialize RS485 UART and Modbus RTU client.	ESP32 can query the VFD slave ID.
3	Read required VFD registers using the INVT mapping document.	Raw values are available for frequency, current, voltage, power/status/fault where supported.
4	Apply scaling and status decoding in VFD driver layer.	Normalized telemetry fields are created.
5	Build JSON exactly according to this guide and the v1.1 specification.	Payload is accepted by cloud schema.
6	POST payload over HTTPS to production endpoint with X-API-Key header.	Cloud returns HTTP 200 and {"ok": true}.
7	Handle error responses and network failures according to retry table.	No data loss during temporary outage.
8	Verify dashboard update and complete FAT checklist.	Firmware is ready for controlled field deployment.

4. System Architecture

The firmware shall implement a layered architecture. The cloud payload must remain stable even if the VFD register map changes. Brand-specific register addresses, scaling, and decoding shall stay inside the VFD driver layer.

```
Solar Pump / Motor
       |
       | Power and control
       v
INVT VFD / Controller
       |
       | RS485 Modbus RTU
       v
AI-Kit Edge Controller
ESP32 + RS485 Transceiver + Local Storage + RTC/Time Sync
       |
       | WiFi / GSM / 4G Internet
       v
HTTPS POST + JSON + X-API-Key
       |
       v
AI-Kit Cloud Platform
FastAPI Ingest API + PostgreSQL + Dashboard + Analytics
```

4.1 Firmware Layering

Layer	Responsibility	Output to Next Layer
Hardware Interface Layer	Configure UART, RS485 direction control, watchdog, power protections, optional GPIOs.	Reliable serial link and device runtime status.
Modbus Driver Layer	Read holding/input registers, handle timeouts, retries, CRC/response validation.	Raw register values or Modbus error state.
VFD Abstraction Layer	Apply VFD_TYPE-specific mapping, scaling, status decoding and fault decoding.	Normalized engineering values.
Telemetry Builder	Create JSON payload with AI-Kit field names.	Validated JSON string.
Local Buffer	Persist telemetry records when upload fails.	FIFO queue of pending records.
Cloud Client	Send HTTPS POST with headers and body; interpret HTTP response.	Success/failure and retry decision.
Configuration Manager	Load and update local configuration without source code modification.	Runtime parameters for all layers.

5. Device Provisioning and Local Configuration

Each controller must be provisioned with a unique Device ID and a unique API key. Credentials must be stored in non-volatile memory and must not be hard-coded in source files committed to repositories.

5.1 Required Local Configuration Parameters

Parameter	Example / Default	Required	Notes
DEVICE_ID	AST-PCB-001	Yes	Unique AI-Kit controller identifier. Must match cloud provisioning record.
API_KEY	Supplied separately	Yes	Device-specific credential. Never print in logs.
FIRMWARE_VERSION	1.0.0	Yes	Reported in each telemetry payload.
VFD_TYPE	INVT	Yes	Selects register map and scaling module.
API_ENDPOINT	https://ai.altiussuryatech.in/api/telemetry/ingest	Yes	Production endpoint.
MODBUS_SLAVE_ID	1	Yes	Must match VFD address.
MODBUS_BAUDRATE	19200	Yes	Recommended default for INVT unless changed at site.
MODBUS_PARITY	E	Yes	Even parity by default.
MODBUS_STOP_BITS	1	Yes	Recommended default.
SAMPLING_INTERVAL_SEC	5	Yes	Telemetry acquisition interval.
UPLOAD_INTERVAL_SEC	30	Yes	Cloud upload interval.
RATED_MOTOR_KW	Site-specific	If deriving power_kw	Required if only power_pct is

			available from VFD.
TIME_SOURCE	NTP / RTC	Recommended	Timestamp accuracy is required for dashboard and analytics.

5.2 Credential Rules

- Never commit API keys to GitHub, firmware repositories, examples, screenshots, logs, or support tickets.
- Never print the API key in serial debug output. If debug is necessary, print only the first 4 and last 4 characters, or mask completely.
- API key rotation must be possible without recompiling firmware.
- Use one API key per physical device. Do not share one key across multiple devices.

6. ESP32 Firmware Workflow

```
Power ON / Reset
   |
   v
Load configuration from NVS / EEPROM / flash configuration area
   |
   v
Initialize watchdog, UART, RS485, local buffer, time source
   |
   v
Connect to WiFi or GSM/4G network
   |
   v
Read Modbus registers from VFD
   |
   v
Apply VFD_TYPE-specific scaling and status/fault decoding
   |
   v
Build telemetry JSON
   |
   v
If network available: POST pending buffered records first, then latest record
If network unavailable: store record in local buffer
   |
   v
Interpret HTTP response
   |
   v
Sleep / wait until next sampling interval
```

6.1 Firmware Tasks

Task	Recommended Period	Failure Handling
Modbus sampling	Every 5 seconds	If Modbus timeout occurs, record communication error and keep last valid values only if explicitly marked.
Cloud upload	Every 30 seconds	Retry with backoff; queue pending record in buffer.
Time synchronization	At boot and periodically	Use RTC fallback if NTP/GSM time is unavailable.
Watchdog service	Within firmware main loop / task scheduler	Reset if communication task hangs.
Configuration validation	At boot	Enter safe error mode if DEVICE_ID or API_KEY is missing.

7. RS485 Modbus Data Acquisition

The ESP32 shall acquire VFD telemetry over RS485 Modbus RTU. INVT-specific addresses, register lengths, scaling and decoding are defined in AST-AIK-FW-002. This API guide defines how normalized values are sent to cloud after register acquisition.

7.1 Recommended RS485 Settings for INVT

Parameter	Recommended Value
Mode	Modbus RTU over RS485
Slave ID	1 unless field configured otherwise
Baud Rate	19200
Parity	Even
Stop Bits	1
Timeout	1000 ms recommended starting point
Retries	3 retries before marking sample as failed
Write Operations	Disabled in this release

7.2 Normalized Fields from VFD Driver

Normalized Field	Source / Derivation
frequency_hz	VFD output frequency after scaling.

motor_voltage_v	VFD motor output voltage after scaling.
motor_current_a	VFD output current after scaling.
power_pct	VFD reported output power percentage after scaling.
power_kw	Direct VFD kW value if available, otherwise derived from power_pct and configured rated motor kW.
run_status	1 when running; 0 when stopped or not running.
fault_code	Decoded VFD fault code or text.
dc_bus_voltage_v	Optional field if available from register map and supported by cloud.
controller_temp_c	Controller or VFD temperature if available.

8. Cloud API Endpoint

Item	Value
Production Endpoint	https://ai.altiussuryatech.in/api/telemetry/ingest
Method	POST
Protocol	HTTPS
TLS	TLS 1.2 or higher
Content Type	application/json
Authentication Header	X-API-Key: <Assigned Device API Key>
Response Format	JSON

8.1 HTTP Request Example

```
POST /api/telemetry/ingest HTTP/1.1
Host: ai.altiussuryatech.in
Content-Type: application/json
X-API-Key: <Assigned Device API Key>

{
  "device_id": "AST-PCB-001",
  "timestamp": "2026-01-01T12:00:00Z",
  "firmware_version": "1.0.0",
  "vfd_type": "INVT",
  "motor_voltage_v": 415.0,
  "motor_current_a": 8.2,
  "power_pct": 51.0,
  "power_kw": 5.1,
  "frequency_hz": 45.0,
  "run_status": 1,
  "fault_code": 0,
  "signal_strength": 64,
  "controller_temp_c": 60.3
}
```

9. Authentication and Security

The AI-Kit Cloud validates the Device ID and X-API-Key before accepting telemetry. The firmware must include the API key in the request header and the device_id in the JSON body.

Security Requirement	Firmware Implementation Rule
Unique Device ID	Store DEVICE_ID in NVS or equivalent non-volatile configuration.
Unique API Key	Store API_KEY securely. Do not hard-code into source repository.
HTTPS only	Use TLS-capable HTTP client. Do not use plain HTTP in production.
No key logging	Do not print API key to serial console or remote logs.
Credential update	Allow field credential replacement without source code modification.
Failed authentication	Do not repeatedly flood server. Back off and report local configuration error.

10. Telemetry Payload

The telemetry payload shall follow the AI-Kit Telemetry Integration Specification v1.1. The firmware may omit unsupported recommended fields only if the cloud backend accepts the missing field and the dashboard can tolerate null values. Mandatory fields must always be sent.

```
{
  "device_id": "AST-PCB-001",
  "timestamp": "2026-01-01T12:00:00Z",
  "firmware_version": "1.0.0",
  "vfd_type": "INVT",
  "pv_voltage_v": 580.2,
  "motor_voltage_v": 415.0,
  "motor_current_a": 8.2,
  "power_pct": 51.0,
  "power_kw": 5.1,
  "frequency_hz": 45.0,
```

```
  "rpm": 1350,
  "flow_lpm": 370,
  "run_status": 1,
  "fault_code": 0,
  "signal_strength": 64,
  "controller_temp_c": 60.3
}
```

10.1 Mandatory and Recommended Fields

Field	Type	Required	Description
device_id	String	Mandatory	Unique AI-Kit Device Identifier.
timestamp	ISO8601 UTC	Mandatory	Telemetry timestamp in UTC.
run_status	Integer	Mandatory	Pump/VFD running status: 1 running, 0 not running.
firmware_version	String	Recommended	Firmware version running on edge controller.
vfd_type	String	Recommended	Configured VFD driver type, e.g., INVT.
pv_voltage_v	Float	Recommended	Solar PV voltage if available.
motor_voltage_v	Float	Recommended	Motor output voltage.
motor_current_a	Float	Recommended	Motor output current.
power_pct	Float	Recommended	VFD output power percentage after scaling.
power_kw	Float	Recommended	Motor power in kW or derived value.
frequency_hz	Float	Recommended	VFD output frequency.
rpm	Float	Optional	Motor speed if available.
flow_lpm	Float	Optional	Water flow rate if flow sensor is installed.
fault_code	Integer/String	Recommended	Decoded VFD/controller fault code.
signal_strength	Integer	Recommended	WiFi/GSM signal indicator.
controller_temp_c	Float	Recommended	Controller temperature or VFD temperature if available.

11. ESP32 WiFi HTTPS Example

The following Arduino-style ESP32 example demonstrates the minimum cloud upload logic. It is intended as implementation guidance and must be adapted to the selected Modbus library, credential storage method and production certificate handling approach.

```
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <HTTPClient.h>

// Development placeholders only. Do not hard-code production credentials.
const char* WIFI_SSID = "<ssid>";
const char* WIFI_PASS = "<password>";
const char* DEVICE_ID = "AST-PCB-001";
const char* API_KEY = "<assigned-device-api-key>";
const char* API_URL = "https://ai.altiussuryatech.in/api/telemetry/ingest";

void connectWiFi() {
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  unsigned long start = millis();
  while (WiFi.status() != WL_CONNECTED && millis() - start < 20000) {
    delay(500);
  }
}

String buildTelemetryJson() {
  // Replace these example values with scaled values from the VFD abstraction layer.
  String json = "{";
  json += "\"device_id\":\"" + String(DEVICE_ID) + "\",";
  json += "\"timestamp\":\"2026-01-01T12:00:00Z\",";
  json += "\"firmware_version\":\"1.0.0\",";
  json += "\"vfd_type\":\"INVT\",";
  json += "\"motor_voltage_v\":415.0,";
  json += "\"motor_current_a\":8.2,";
  json += "\"power_pct\":51.0,";
  json += "\"power_kw\":5.1,";
  json += "\"frequency_hz\":45.0,";
  json += "\"run_status\":1,";
  json += "\"fault_code\":0,";
  json += "\"signal_strength\":" + String(WiFi.RSSI()) + ",";
  json += "\"controller_temp_c\":60.3";
  json += "}";
  return json;
}

bool postTelemetry(const String& payload) {
  if (WiFi.status() != WL_CONNECTED) return false;

  WiFiClientSecure client;
  // For production, prefer client.setCACert(root_ca). setInsecure() is for development only.
  client.setInsecure();

  HTTPClient http;
  if (!http.begin(client, API_URL)) return false;

  http.addHeader("Content-Type", "application/json");
  http.addHeader("X-API-Key", API_KEY);

  int code = http.POST(payload);
  String response = http.getString();
  http.end();

  Serial.printf("HTTP %d\n", code);
  Serial.println(response);

  return (code == 200);
```

```
}

void setup() {
  Serial.begin(115200);
  connectWiFi();
}

void loop() {
  String payload = buildTelemetryJson();
  bool ok = postTelemetry(payload);
  if (!ok) {
    // Store payload in local circular buffer and retry later.
  }
  delay(30000);
}
```

11.1 Production Notes for ESP32 Example

- Replace hard-coded placeholders with values loaded from NVS or a secure local configuration mechanism.
- Replace setInsecure() with root CA certificate validation before production deployment where feasible.
- Use a JSON library such as ArduinoJson for robust payload construction instead of string concatenation in production code.
- Add watchdog protection, retry backoff and local buffering before field deployment.
- Ensure timestamp is current UTC time from NTP, GSM network time, or RTC.

12. TinyGSM / 4G Implementation Notes

For GSM/4G deployments, the same HTTPS endpoint, headers and JSON payload shall be used. The only difference is the network transport layer. The firmware developer may use modules such as SIM7600, A7670, SIM7080 or equivalent based on field network availability.

Item	Guidance
APN	Use SIM provider APN. Make this configurable.
Network Time	Prefer modem/network time or NTP after data connection.
TLS	Verify module and library support for HTTPS/TLS.
Signal Strength	Report signal_strength in telemetry when available.
Reconnect	If PDP/data context fails, reconnect before retrying upload.
Power	4G modules need stable supply with adequate peak current. Brownouts cause intermittent failures.

```
// TinyGSM implementation outline only
// 1. Initialize modem serial port
// 2. Restart modem
// 3. Connect to network
// 4. Open GPRS/LTE data context using APN
// 5. Create HTTPS client
// 6. POST same JSON payload to https://ai.altiussuryatech.in/api/telemetry/ingest
// 7. Add headers: Content-Type and X-API-Key
// 8. Interpret HTTP response exactly as WiFi implementation
```

13. Retry Logic and Local Buffering

The firmware shall continue collecting telemetry during communication outages. Failed records shall be written to a local circular buffer and uploaded automatically when connectivity is restored.

13.1 Required Buffer Behavior

- Minimum capacity: 1000 telemetry records.
- Recommended implementation: non-volatile circular FIFO buffer.
- On network failure, enqueue the record and continue sampling.
- On reconnect, upload oldest pending records first, then latest record.
- Delete buffered record only after HTTP 200 success response.
- Avoid duplicate uploads wherever practical. A local sequence number is recommended even if not sent to cloud in v1.0.

13.2 Retry Backoff

Failure Type	Recommended Action
WiFi/GSM not connected	Do not attempt HTTP. Store record locally and reconnect network.
DNS failure	Retry after backoff. Store record.
TLS handshake failure	Check certificate/time; retry after backoff.
HTTP 500/502/503/504	Server-side or gateway issue. Store and retry later.
HTTP 401/403	Do not aggressively retry. Mark authentication/provisioning error and use long backoff.
HTTP 400	Payload is invalid. Log locally; do not repeatedly resend the same malformed payload.

14. Cloud Response Handling

HTTP Code	Typical Response	Firmware Action
200	{"ok": true}	Success. Delete uploaded record from local buffer. Continue normal operation.
400	{"detail": "missing device_id"}	Invalid payload. Keep diagnostic log without API key. Fix firmware payload. Do not flood server.
401	{"detail": "missing api key"}	API key missing from header. Check credential storage and header creation.
403	{"detail": "invalid device or api key"}	Wrong Device ID/API key pair or unprovisioned device. Stop rapid retries and request provisioning check.
500 or timeout	Internal Server Error / no response	Store record and retry later with backoff.

15. Timing, Timestamp and Data Quality Rules

- Telemetry timestamp shall use ISO8601 UTC format, for example 2026-01-01T12:00:00Z.
- Sampling interval target is 5 seconds unless changed by approved configuration.
- Cloud upload interval target is 30 seconds unless changed by approved configuration.
- If the timestamp is not valid at boot, firmware shall synchronize time before uploading or mark data according to agreed future rules.
- Do not send impossible values without error marking. For example, negative current, impossible frequency, or invalid scaled data should be flagged locally.
- When Modbus read fails, avoid silently reusing old values as if they are fresh. If last-good values are used, firmware should internally mark communication quality for future telemetry expansion.

16. Verification Using cURL

Before firmware testing, the cloud endpoint can be verified using cURL from a computer. Replace the API key with the assigned key from the restricted provisioning record. Do not paste real keys into public logs or chat messages.

```
curl -i \
  -H "Content-Type: application/json" \
  -H "X-API-Key: <Assigned Device API Key>" \
  -d '{
    "device_id":"AST-PCB-001",
    "timestamp":"2026-01-01T12:00:00Z",
    "firmware_version":"1.0.0",
    "vfd_type":"INVT",
    "motor_voltage_v":415.0,
    "motor_current_a":8.2,
    "power_pct":51.0,
    "power_kw":5.1,
    "frequency_hz":45.0,
    "run_status":1,
    "fault_code":0,
    "signal_strength":64,
    "controller_temp_c":60.3
  }' \
  https://ai.altiussuryatech.in/api/telemetry/ingest
```

16.1 Expected Success

```
HTTP/1.1 200 OK
Content-Type: application/json

{"ok": true}
```

17. Firmware Acceptance Test Checklist

The firmware shall not be accepted for field deployment until the following tests are completed and signed off.

No	Test	Expected Result	Pass/Fail	Remarks
1	ESP32 boots and firmware version is visible in debug/service screen.	Firmware starts without crash.		
2	Device ID is loaded from non-volatile configuration.	Correct assigned DEVICE_ID is used.		
3	API key is loaded but not printed in logs.	Credential is protected.		
4	RS485 communication with VFD works.	Modbus response received from configured slave ID.		
5	INVT registers are read according to mapping document.	Raw values match VFD display or known reference.		
6	Scaling is correct.	frequency_hz, current, voltage, power_pct are		

		reasonable.		
7	run_status is decoded correctly.	Run/stop state matches actual VFD state.		
8	fault_code is decoded correctly.	Known fault or no-fault condition is represented correctly.		
9	Telemetry JSON is valid.	JSON parses and includes mandatory fields.		
10	HTTPS upload works.	Cloud returns HTTP 200 and {"ok": true}.		
11	Dashboard updates.	Latest timestamp and values appear on AI-Kit dashboard.		
12	Network outage test.	Records are buffered locally during outage.		
13	Network recovery test.	Buffered records upload automatically after network returns.		
14	Power cycle test.	Device resumes operation and retains configuration.		
15	Wrong API key test.	Cloud rejects with 403; firmware does not flood server.		
16	Missing API key test.	Cloud rejects with 401; firmware reports configuration error.		
17	Watchdog/recovery behavior.	Firmware recovers from communication hang.		
18	Final signoff.	Firmware ready for controlled deployment.		

18. Commissioning Checklist

Stage	Checklist Item	Status
Before Power-On	Verify wiring polarity for RS485 A/B and power supply rating.	
Before Power-On	Confirm VFD Modbus slave ID, baud rate, parity and stop bits.	
Before Power-On	Confirm Device ID and API key match provisioning record.	
Network	Confirm WiFi credentials or SIM/APN configuration.	
Network	Confirm internet access from installation location.	
Firmware	Confirm firmware version and VFD_TYPE = INVT.	
Modbus	Confirm live frequency/current/voltage reads are reasonable.	
Cloud	Confirm HTTP 200 from telemetry ingest endpoint.	
Dashboard	Confirm live timestamp updates.	
Reliability	Perform short internet disconnect and reconnect test.	
Documentation	Record firmware version, device ID, site, date and commissioning engineer.	

19. Troubleshooting Guide

Symptom	Likely Cause	Action
No Modbus response	Wrong slave ID, baud/parity mismatch, A/B reversed, no VFD power.	Check RS485 wiring, VFD communication parameters and termination.
Frequency reads wrong	Scaling mismatch or wrong register.	Check INVT register mapping and scaling factor.
HTTPS fails	No internet, TLS issue, DNS issue, wrong endpoint.	Check network, timestamp, certificate handling and URL.
HTTP 401	Missing X-API-Key header.	Verify header creation and API key storage.
HTTP 403	Invalid Device ID/API key pair or device not provisioned.	Verify restricted provisioning record.
HTTP 400	Malformed JSON or missing mandatory field.	Validate JSON and mandatory fields.
Dashboard not updating	Upload failed, timestamp old, wrong device ID, backend issue.	Check API response and dashboard latest data.
Works on WiFi but not GSM	APN, signal, SIM data, TLS support or power issue.	Check modem logs, signal and supply current.
Device resets during upload	Power supply brownout, modem peak current, watchdog.	Improve power design and inspect reset reason.
Duplicate records	Buffer deletion logic incorrect.	Delete only after HTTP 200 and consider local sequence ID.

20. Handover Requirements

The firmware developer shall provide the following items before firmware is accepted as ready for controlled field deployment.

- Compiled firmware binary and source code version tag.
- Firmware architecture description and library list.
- Configuration method for Device ID, API key, VFD_TYPE and Modbus parameters.
- Evidence of successful Modbus register reads.
- Evidence of successful HTTPS telemetry upload to AI-Kit Cloud.
- Screenshot or log showing HTTP 200 response without exposing API key.
- Dashboard screenshot showing latest telemetry update.
- Completed Firmware Acceptance Test checklist.
- Known limitations and pending issues.
- Recommended next firmware version improvements.

Appendix A. Sample Payload


```
{
  "device_id": "AST-PCB-001",
  "timestamp": "2026-01-01T12:00:00Z",
  "firmware_version": "1.0.0",
  "vfd_type": "INVT",
  "pv_voltage_v": null,
  "motor_voltage_v": 415.0,
  "motor_current_a": 8.2,
  "power_pct": 51.0,
  "power_kw": 5.1,
  "frequency_hz": 45.0,
  "rpm": null,
  "flow_lpm": null,
  "run_status": 1,
  "fault_code": 0,
  "signal_strength": 64,
  "controller_temp_c": 60.3
}
```


Appendix B. Firmware State Machine

```
STATE_BOOT
  -> Load configuration
  -> Validate required parameters
  -> STATE_NETWORK_CONNECT

STATE_NETWORK_CONNECT
  -> Connect WiFi/GSM
  -> Sync time
  -> STATE_MODBUS_READ

STATE_MODBUS_READ
  -> Read VFD registers
  -> If success: STATE_BUILD_PAYLOAD
  -> If failure: record Modbus error and continue per configuration

STATE_BUILD_PAYLOAD
  -> Apply scaling
  -> Build JSON
  -> STATE_UPLOAD

STATE_UPLOAD
  -> If network available: POST telemetry
  -> If HTTP 200: mark success and flush buffer
  -> If failure: STATE_BUFFER_RECORD

STATE_BUFFER_RECORD
  -> Store record in circular buffer
  -> STATE_WAIT

STATE_WAIT
  -> Feed watchdog
  -> Wait until next sample/upload interval
  -> STATE_MODBUS_READ
```

Appendix C. Developer Notes

- Use a stable JSON builder library for production firmware.
- Add compile-time and runtime firmware version visibility.
- Keep VFD driver code modular so Fuji or generic Modbus support can be added later without changing cloud payload fields.
- Do not implement remote VFD writes unless explicitly approved in a future controlled release.

- Field diagnostics should be useful but must never expose API keys.
- For GSM deployments, validate modem power supply under transmit conditions.
- For TLS, certificate validation is preferred. setInsecure() should be treated as a development shortcut only.

Approval

Role	Name	Signature / Date
Product Owner	K N Vishwanath	
Firmware Developer		
Test / Commissioning Engineer		