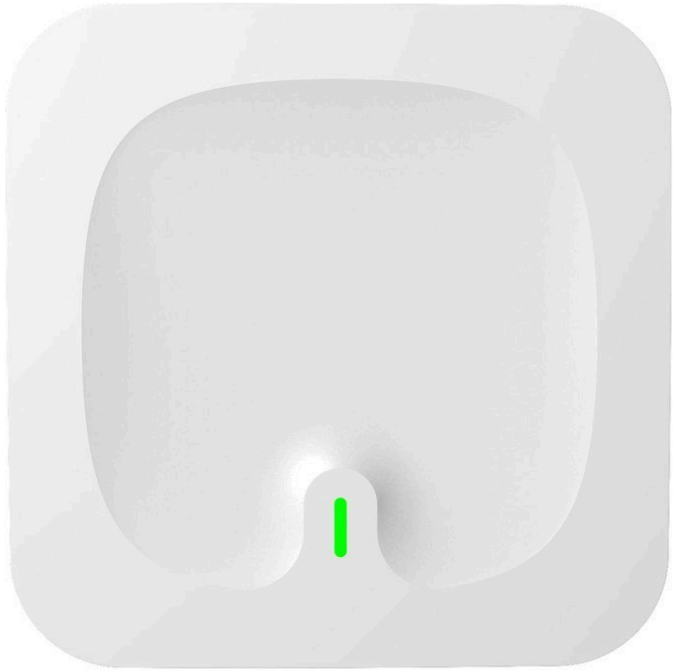
INTEGRATION GUIDE

AoA Location Engine

Receive AoA Tag Positions in Your Own System

Identium Location Engine with ID-AOA-GW25 anchors: architecture, coordinates and timing, the platform's REST API, WebSocket and alert webhook, the standalone engine feeds, field reference and tested example consumers in Python and Node.js



THIS MANUAL COVERS	VERSION	PLATFORM
Identium RTLS platform API (REST, WebSocket, webhook)	4.19	HTTP · JSON
Identium Location Engine standalone feeds	–	WebSocket · UDP · HTTP · MQTT
AoA Integration Kit (example consumers)	1.0	Python · Node.js

Document version 1.0 · November 2026

Identium Tech Solutions Pvt. Ltd.
Plot No. 5, First Floor, Raja Dhirsain Marg, Garhi, Sant Nagar, East of Kailash, New Delhi 110065, India
+91 97181 99516 · 011-4714 7839 · WhatsApp +91 82877 32647 · info@identium.in

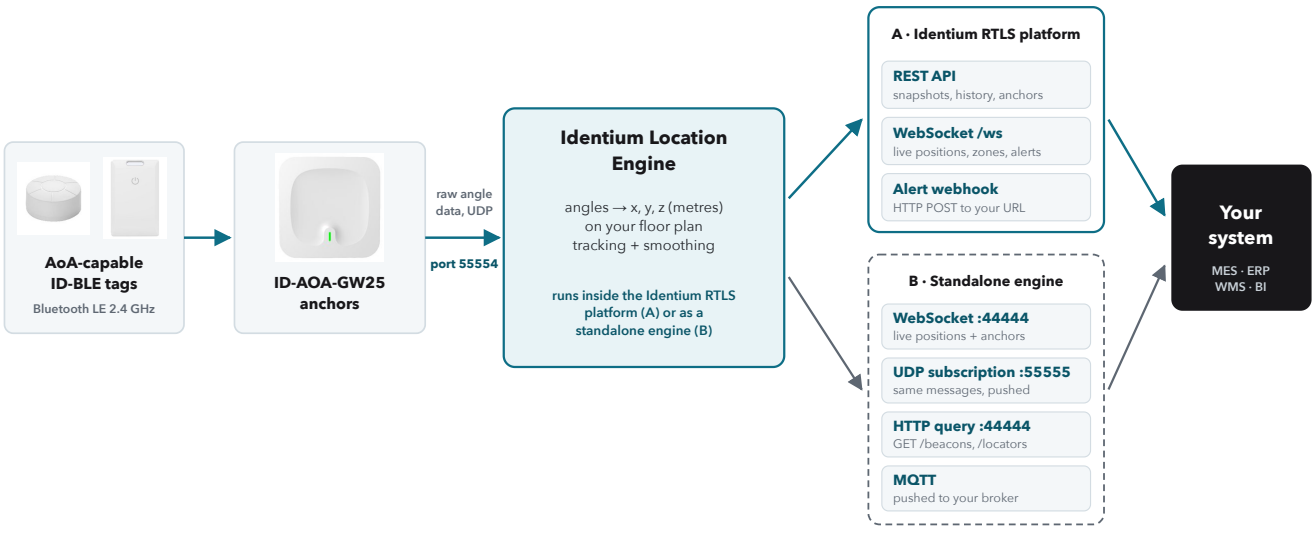
www.identiumone.com · www.indiarfidstore.com
Creating identities for all

ID-AOA-GW25 anchors and the Identium Location Engine locate AoA-capable ID-BLE tags in metres on your floor plan. This guide is for the developer who brings those positions into another system: what the engine outputs and how often, the coordinate system, the Identium RTLS platform's REST API, live WebSocket and alert webhook, the standalone engine's feeds, every field, sample messages, and example consumers in Python and Node.js that you can run as they are.

CONTENTS

01/ How It Fits Together
02/ What Goes In and What Comes Out
03/ Coordinate System and Units
04/ Update Rate and Timing
05/ Option A: Sign In and REST API
06/ Option A: Live WebSocket
07/ Option A: Alert Webhook
08/ Option B: Standalone Engine Feeds
09/ Field Reference
10/ Example Consumers
11/ Configuration Reference
12/ Troubleshooting
13/ Support

01/ How It Fits Together



AoA tags → ID-AOA-GW25 anchors → Identium Location Engine → your system, through the Identium RTLS platform (A) or the standalone engine's feeds (B).

AoA-capable ID-BLE tags broadcast; each ID-AOA-GW25 anchor measures the direction every broadcast arrives from and streams that raw measurement to the **Identium Location Engine**. The engine combines it with each anchor's stored position, height and orientation and produces a **position per tag in metres** on your floor plan. Your software takes those positions from one of two places:

	Use it when	You get
A · Identium RTLS platform API	The site runs the Identium RTLS platform (on site or hosted by Identium). This is the usual choice.	REST snapshots and history, a live WebSocket, an alert webhook. Positions come with tag names, floors, zone events and alerts.
B · Standalone engine feeds	You run your own RTLS software and need only the position stream. Identium sets up a standalone engine for the project.	WebSocket, UDP and HTTP feeds (MQTT on request) with positions on the engine's map and the anchors' status.

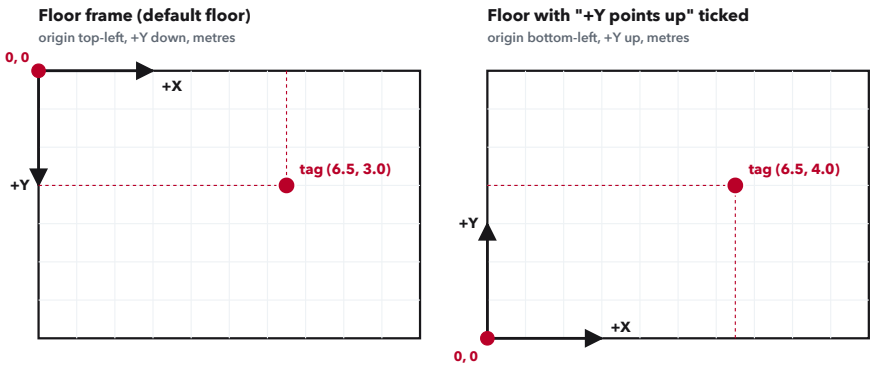
> **NOTE**
> The anchors' own stream is binary and is read only by the engine. Integrate with the engine's outputs in this guide, never with the anchors directly.

02/ What Goes In and What Comes Out

Stage	Data	Notes
Tag → anchor	Bluetooth LE broadcast with a direction-finding tone	Once per broadcast interval of the tag
Anchor → engine	One UDP message (port 55554) per broadcast heard: the raw antenna measurement, the anchor's MAC and the tag's MAC	Binary; the engine polls each anchor every 2 s to keep the stream running
Engine → your system	Per tag: **x, y, z** in metres on a floor or map, an accuracy estimate, a time; plus the anchor list and status	Formats in sections 05–09

With one anchor above a tag the engine fixes the point on a plane at the site's **tag height**; where two or more anchors hear the tag it combines them. It then tracks each tag: a resting tag is held still, and a move is published once several fixes agree. For planning, accuracy and anchor set-up see the *ID-AOA-GW25 User Manual*.

03/ Coordinate System and Units



Same tag, same place:

the floor is 7 m deep, so
y = 3.0 from the top is
y = 4.0 from the bottom.

z = height of the tag above the floor (the configured tag height for one-anchor fixes).

The standalone engine (B) uses the bottom-left frame of its own map (mapId).

The same tag in the two floor frames the platform supports. Positions are always in metres from the floor's origin.

Value	Unit and format	Notes
x, y	metres	A: in the floor's own frame, origin top-left with +Y down (default) or bottom-left with +Y up when the floor's `origin_bottom` is true. B: the engine map's frame, origin bottom-left, +Y up.
z	metres	Height of the tag above the floor; for one-anchor fixes it is the site's set tag height.
accuracy_meters	metres	The engine's estimate of the uncertainty: a guide, not a guarantee. Never below 0.4 m.
Times	A: `computed_at` ISO 8601 UTC; `ts` and `t` Unix milliseconds. B: `updatedAt`, `calculatedAt`, `timestamp` Unix milliseconds	Use these, not the time a message arrives.
Tag MAC	A: `C3:10:00:00:00:02` (upper case, colons). B: `c31000000002` (lower case, no colons)	Normalise before you join A and B data.
Floor / map	A: `floor_id` (integer, from `GET /api/ble/floors`). B: `mapId` (string)	
Anchor angles	degrees	Azimuth 0 = LED / network-port side toward −Y of the plan, clockwise; elevation 0 = level ceiling mount.

04/ Update Rate and Timing

Item	Value
Engine publish rate	Once a second per tag, while the tag has given a usable fix in the last 15 s
WebSocket (A)	Batched messages; at most one update per tag every 0.4 s, so in practice about one per second per tag
History (A)	At most one stored position per tag per second
Resting tag	Held still: the published point moves only when the tag is confirmed to have moved (0.3 m dead band)
Moving tag	Shown at its new place within about 10 s of being picked up; steps of 1–2 m while carried
Tag no longer heard	Not published after 15 s; REST keeps the last known position with its `computed_at`
What limits the rate	The tag's broadcast interval: in Identium's tests about one broadcast in three gave a usable angle. Set AoA tags to 1 s or faster.

> **NOTE**
> Positions reach your system **within seconds**. Use them for location, presence, zones and search, and use the time stamps to decide whether a position is still current.

05/ Option A: Sign In and REST API

Base URL is the address you open the platform at, for example `http://192.168.1.20:3004`, or `https://rtls.example.com/path` when it is published under a path. Ask your administrator to create a user for the integration under **User Management**; every API call and the WebSocket need a signed-in session.

1. `POST /api/auth/login` with `{"username": "…", "password": "…"}`. The reply sets a session cookie (`connect.sid`).
2. Send that cookie with every request and with the WebSocket. A session lasts 24 hours.
3. A reply of **401** `{"success": false, "message": "Not authenticated"}` means the session has ended (expired, or the server restarted): sign in again.

```
curl -c jar.txt -H 'Content-Type: application/json' \
     -d '{"username":"api-reader","password":"…"}' http://192.168.1.20:3004/api/auth/login
curl -b jar.txt http://192.168.1.20:3004/api/ble/positions
```

Sign in and read the last known positions with curl.

Request	Returns
`GET /api/ble/positions` (`?floor_id=`)	Last known position of every tag, newest first, with tag name and asset type. Keep rows with `source` = `ble_aoa`.
`GET /api/ble/positions/{mac}/history`	Stored positions (≤ 1 per second). `since`, `until` (Unix ms; default the last hour), `limit` (default 600, max 5000), `order=desc`.
`GET /api/ble/positions/{mac}/trail`	The last 30 points, kept in memory (cleared on restart).
`GET /api/ble/locate?q=`	Search by tag name, MAC, type or owner: last position with `age_seconds` and `status`.

Request	Returns
`GET /api/ble/floors`	Floors: `id`, `name`, `width_meters`, `height_meters`, `ceiling_m`, `origin_bottom`.
`GET /api/ble/locators`	Anchors: MAC, name, floor, x, y, z, azimuth, elevation, rotation, `online`, `last_seen`.
`GET /api/ble/beacons`	Registered tags: MAC, name, asset type, owner.
`GET /api/ble/zones`, `/zones/{id}/occupants`	Zones (rectangle or polygon, metres) and the tags in a zone now.
`GET /api/ble/events`	Zone enter / leave history.
`GET /api/ble/alerts`, `POST /api/ble/alerts/{id}/ack`	Alerts, and acknowledging one.
`GET /api/ble/aoa/status`	Engine health: anchors heard, tags heard, frames received and decoded.


```
{
 "success": true,
 "positions": [
  {
   "beacon_mac": "C3:10:00:00:00:02",
   "floor_id": 1,
   "x_meters": 11.004,
   "y_meters": 1.602,
   "z_meters": 0.75,
   "accuracy_meters": 1,
   "gateway_count": 0,
   "source": "ble_aoa",
   "zone_ext_id": null,
   "sos": 0,
   "computed_at": "2026-11-20T13:03:59.000Z",
   "beacon_name": "Laptop QA-12",
   "asset_type": "Asset",
   "battery": null,
   "owner": null
  }
 ]
}
```


GET /api/ble/positions: one AoA row (the list holds every tag).

```
{
 "success": true,
 "history": [
  {
   "x_meters": 10.98,
   "y_meters": 1.573,
   "z_meters": 0.75,
   "accuracy_meters": 1,
   "gateway_count": 0,
   "source": "ble_aoa",
   "floor_id": 1,
   "t": 1795179838000
  },
  {
   "x_meters": 11.004,
   "y_meters": 1.602,
   "z_meters": 0.75,
   "accuracy_meters": 1,
   "gateway_count": 0,
   "source": "ble_aoa",
   "floor_id": 1,
   "t": 1795179839000
  }
 ],
 "since": 1795176239000,
 "until": 1795179839000,
 "total": 3600,
 "truncated": true
}
```


GET /api/ble/positions/C3:10:00:00:00:02/history?limit=2&order=desc

06/ Option A: Live WebSocket

Open `ws://<base>/ws` (`wss://` for an https address) with the session cookie in the `Cookie` header. The platform pushes JSON text messages `{"type": …, "data": …}` as things happen; you send nothing. Without a valid cookie the connection is refused with HTTP 401.

type	When	data
`ble_positions`	About once a second while tags are located	`source` (`ble_aoa` for AoA), `positions[]`, `ts`
`ble_zone_event`	A tag enters or leaves a zone, or stays past the zone's dwell limit	`zone_id`, `zone_name`, `zone_type`, `beacon_mac`, `floor_id`, `event_type` (`enter`, `leave`, `dwell`), `alert`, `ts`
`ble_alert`	An alert is raised	The alert object (section 07)
`ble_locators`	An anchor comes online, goes offline or is edited	`locators[]` with `mac`, `online` and, when edited, its position and orientation

- The same socket also carries updates for gateways and active RFID readers. **Ignore types you do not use.**

- Filter `ble_positions` on `data.source == "ble_aoa"`; gateway (RSSI) positions use the same type with other sources.
- `beacon_name` in zone events can be null: look the name up once from `GET /api/ble/beacons`.
- If the socket closes, wait a few seconds, sign in again and reconnect. Read `GET /api/ble/positions` after a reconnect to catch up.


```
{
 "type": "ble_positions",
 "data": {
  "source": "ble_aoa",
  "positions": [
   {
    "beacon_mac": "C3:10:00:00:00:01",
    "floor_id": 1,
    "x": 5.19,
    "y": 4.398,
    "z": 0.75,
    "accuracy_meters": 1,
    "gateway_count": 0,
    "source": "ble_aoa",
    "battery": null,
    "sos": false,
    "zone_ext_id": null
   },
   {
    "beacon_mac": "C3:10:00:00:00:03",
    "floor_id": 1,
    "x": 15.6,
    "y": 6.0,
    "z": 0.75,
    "accuracy_meters": 1,
    "gateway_count": 0,
    "source": "ble_aoa",
    "battery": null,
    "sos": false,
    "zone_ext_id": null
   }
  ],
  "ts": 1795179851785
 }
}
```


ble_positions (two tags)

```
{
 "type": "ble_zone_event",
 "data": {
  "zone_id": 1,
  "zone_name": "Corridor east",
  "zone_type": "alert",
  "beacon_mac": "C3:10:00:00:00:03",
  "floor_id": 1,
  "beacon_name": null,
  "event_type": "enter",
  "alert": true,
  "ts": 1795179851785
 }
}
```

ble_zone_event

07/ Option A: Alert Webhook

The platform can also POST every alert to one URL of yours, so your system hears about alerts without keeping a WebSocket open. Identium sets the URL, and an optional shared token, on the server.

Request	`POST` to your URL, `Content-Type: application/json`
Token	Header `x-alert-token: <token>` when a token is set. Reject posts without it.
Body	One alert object (below); the same object is pushed on the WebSocket as `ble_alert`
Your reply	Any 2xx. Each alert is posted once; read `GET /api/ble/alerts` after an outage of your receiver.
Repeats	The same tag, alert type and zone raise at most one alert a minute

alert_type	Raised when
`zone_breach`	A tag enters a zone set to alert on enter, or leaves one set to alert on leave
`zone_dwell`	A tag stays in a zone longer than the zone's dwell limit
`tag_offline`	A registered tag has not been located for the offline period (10 minutes by default)
`sos`	A tag with an alarm button reports it pressed

```
{
 "id": 1,
 "acknowledged": 0,
 "created_at": "2026-11-20T13:02:05.556Z",
 "alert_type": "zone_breach",
 "beacon_mac": "C3:10:00:00:00:03",
 "beacon_name": "Visitor badge 21",
 "zone_id": 1,
 "zone_name": "Corridor east",
 "floor_id": 1,
 "message": "Visitor badge 21 entered Corridor east"
}
```

Webhook body, as received from the platform

```
#!/usr/bin/env python3
# Receives the Identium RTLS platform's alert webhook (Python 3, standard library only).
#   python aoa_webhook_receiver.py 8080 <token>      -> URL to give Identium: http://<this-
pc>:8080/aoa-alerts
import http.server, json, sys

PORT = int(sys.argv[1]) if len(sys.argv) > 1 else 8080
TOKEN = sys.argv[2] if len(sys.argv) > 2 else ''

class Receiver(http.server.BaseHTTPRequestHandler):
    def do_POST(self):
        body = self.rfile.read(int(self.headers.get('Content-Length', 0)))
        if TOKEN and self.headers.get('x-alert-token') != TOKEN:
            self.send_response(403); self.end_headers(); return
        alert = json.loads(body)
        print(f"{alert['created_at']}  {alert['alert_type']:<12} {alert.get('beacon_mac')}
{alert.get('message')}")
        self.send_response(200); self.end_headers(); self.wfile.write(b'ok')
    def log_message(self, *args):
        pass

http.server.HTTPServer(('0.0.0.0', PORT), Receiver).serve_forever()
```

kit/python/aoa_webhook_receiver.py: a receiver with no extra packages

08/ Option B: Standalone Engine Feeds

Where the Identium Location Engine runs as a standalone service for your project, Identium can open its direct feeds to your system. They carry positions on the engine's own map (origin bottom-left, +Y up, metres) and the anchors' status. Tag names, zones and alerts then live in your system.

Feed	Address	How it works
WebSocket	`ws://<engine>:44444`	Connect and receive frames as positions are calculated. Frames are compressed (zlib).
UDP subscription	UDP `<engine>:55555`	Send `subscribe:raw` (plain JSON) or `subscribe:zlib` / `subscribe:gzip` (compressed) and **repeat it every 30 s**; send `unsubscribe` to stop. One subscriber by default.
HTTP query	`http://<engine>:44444`	`GET /beacons` (optional `?macs=a,b`), `GET /beacons/{mac}`, `GET /locators`, `GET /locators/{mac}`: the latest values as JSON; 404 for an unknown MAC.
MQTT	Your broker	On request: the engine publishes the same messages, compressed, to your broker. Broker, topic and credentials are agreed with Identium.

CAUTION

As installed, the engine accepts connections only from its own computer. Ask Identium to open the feeds you need to your network, and keep them on a private network or VPN: they carry no sign-in.

Every message is `{"type": …, "data": …, "timestamp": <ms>}`. A message that does not start with `{` is compressed: inflate it with zlib (or gzip) first. `sensors` carries the tags, keyed by tag MAC; `locators` carries the anchors, keyed by anchor IP address.


```
{
 "type": "sensors",
 "timestamp": 1795179812720,
 "data": {
  "c31000000002": {
   "x": 10.993,
   "y": 10.382,
   "z": 0.75,
   "rssi": -58.2,
   "mapId": "gf01",
   "zoneId": null,
   "gps": null,
   "lastGateway": "3c001d250007",
   "nearestGateway": "3c001d250007",
   "updatedAt": 1795179812718,
   "calculatedAt": 1795179812718
  }
 }
}
```


sensors: tag positions (one tag shown)

```
{
 "type": "locators",
 "timestamp": 1795179815720,
 "data": {
  "192.168.10.107": {
   "mac": "3c:00:1d:25:00:07",
   "online": true,
   "updatedAt": 1795179815719,
   "angles": [],
   "info": {
    "name": "Lab",
    "type": "AOA",
    "mapId": "gf01",
    "x": 11.75,
    "y": 9.5,
    "z": 3.0,
    "azimuth": 0,
    "elevation": 0,
    "rotation": 0,
    "LAN": {
     "ip": "192.168.10.107",
     "mode": "dhcp"
    }
   }
  }
 }
}
```


locators: anchors (one anchor shown)

09/ Field Reference

A · Position (REST row and WebSocket item)

Field (REST / WebSocket)	Type	Meaning
`beacon_mac`	string	Tag MAC, `AA:BB:CC:DD:EE:FF`
`floor_id`	integer	Floor the position is on
`x_meters` / `x`, `y_meters` / `y`	number, m	Position in the floor's frame (section 03)
`z_meters` / `z`	number, m	Tag height above the floor
`accuracy_meters`	number, m	Uncertainty estimate, 0.4 m minimum
`source`	string	`ble_aoa` for AoA; other values are gateway (RSSI) or active RFID positions
`gateway_count`	integer	0 for AoA
`zone_ext_id`	string or null	Zone of the engine's own map, when one is set; platform zones arrive as zone events
`sos`	0/1 or boolean	Alarm button pressed, for tags with a button; false otherwise
`battery`	number or null	Null for most AoA tags; treat any value as indicative
`computed_at` (REST)	ISO 8601 UTC	When the position was last updated
`beacon_name`, `asset_type`, `owner` (REST)	string or null	From the tag's registration
`ts` (WebSocket, on `data`)	Unix ms	When the batch was sent

A · Zone event and alert

Field	Meaning
`zone_id`, `zone_name`, `zone_type`	The zone, as set up under Geo-fence Zones
`event_type`	`enter`, `leave` or `dwell`
`alert`	True when the zone is set to raise an alert for this event
`id`, `alert_type`, `message`	Alert number, type (section 07) and readable text
`acknowledged`, `created_at`	0 until acknowledged; time raised (ISO 8601 UTC)

B · Tag in `sensors`

Field	Type	Meaning
key	string	Tag MAC, lower case, no colons
`x`, `y`, `z`	number, m	Position on the engine's map (origin bottom-left, +Y up)
`mapId`, `zoneId`	string	Map, and the engine's zone (null when none)
`rssi`	number, dBm	Signal strength of the tag at the anchors
`lastGateway`, `nearestGateway`	string	MAC (no colons) of the anchor that last heard the tag, and of the nearest anchor
`updatedAt`, `calculatedAt`	Unix ms	Last heard; when the position was calculated
`gps`	null or {lat, lon}	Only when the map is anchored to GPS coordinates
`userData`, `userDataTs`	object	Optional status block from tags that send one; contents depend on the tag model

B · Anchor in `locators`

Field	Meaning
key	Anchor IP address
`mac`, `online`, `updatedAt`	Anchor MAC (lower case, colons), whether the engine hears it, last update (Unix ms)
`info.name`, `info.type`, `info.mapId`	Name, `AOA`, map
`info.x`, `info.y`, `info.z`	Anchor position on the map, metres
`info.azimuth`, `info.elevation`, `info.rotation`	Mounting angles in degrees
`info.LAN`	`ip`, `mode` (`dhcp` or `static`), and the subnet mask and gateway when set

10/ Example Consumers

The **Identium AoA Integration Kit** on the Resource Hub holds these programs. They were tested on 20 November 2026 against the Identium RTLS platform 4.19 (sign-in, REST, WebSocket, webhook) and against the standalone feed formats in section 08, with Python 3.14 + websocket-client 1.9 and Node.js 25 + ws 8. Run them as they are, then copy the parts you need.

Python: platform positions, snapshot + live

```
#!/usr/bin/env python3
# Identium RTLS platform: AoA tag positions over REST (snapshot) and WebSocket (live).
#   pip install websocket-client
#   python aoa_platform.py http://192.168.1.20:3004 <username> <password>
import json, sys, time, urllib.request, http.cookiejar
import websocket                        # pip install websocket-client

BASE = (sys.argv[1] if len(sys.argv) > 1 else 'http://localhost:3004').rstrip('/')
USER = sys.argv[2] if len(sys.argv) > 2 else 'api-reader'
PASSWORD = sys.argv[3] if len(sys.argv) > 3 else ''

def sign_in():
    """Sign in and return (opener, cookie header). The session cookie authorises REST and
WebSocket."""
    jar = http.cookiejar.CookieJar()
    op = urllib.request.build_opener(urllib.request.HTTPCookieProcessor(jar))
    req = urllib.request.Request(BASE + '/api/auth/login', method='POST',
                                 data=json.dumps({'username': USER, 'password': PASSWORD}).encode(),
                                 headers={'Content-Type': 'application/json'})
    with op.open(req, timeout=10) as r:
        if not json.load(r).get('success'):
            sys.exit('sign-in failed')
    return op, '; '.join(f'{c.name}={c.value}' for c in jar)

def get(op, path):
    with op.open(BASE + path, timeout=10) as r:
        return json.load(r)

def on_message(ws, raw):
    m = json.loads(raw)
    kind, d = m.get('type'), m.get('data') or {}
    if kind == 'ble_positions' and d.get('source') == 'ble_aoa':
        for p in d['positions']:
            print(f"pos   {p['beacon_mac']}  floor {p['floor_id']}  x={p['x']:.2f}  y={p['y']:.2f} m"
                  f"  ±{p['accuracy_meters']} m  zone={p.get('zone_ext_id')}")
    elif kind == 'ble_zone_event':
        print(f"zone  {d['beacon_mac']}  {d['event_type']:<5} {d.get('zone_name')}  alert=
{d.get('alert')}")
    elif kind == 'ble_alert':
        print(f"alert {d.get('alert_type')}: {d.get('message')}")

while True:                             # sign in again after a server restart or network loss
    try:
        op, cookie = sign_in()
        for p in get(op, '/api/ble/positions')['positions']:      # 1. snapshot: last known position
            if p['source'] == 'ble_aoa':
                print(f"last  {p['beacon_mac']}  {p.get('beacon_name') or '-':<16} floor
{p['floor_id']}"
                      f"  x={p['x_meters']:.2f}  y={p['y_meters']:.2f} m  at {p['computed_at']}")
        ws_url = 'ws' + BASE[4:] + '/ws'                          # 2. live: http -> ws, https ->
wss
        websocket.WebSocketApp(ws_url, header=[f'Cookie: {cookie}'],
                               on_message=on_message).run_forever(ping_interval=30)
    except Exception as e:
        print('connection problem:', e)
    time.sleep(5)
```

kit/python/aoa_platform.py. Needs Python 3.8 or later and `pip install websocket-client`; run `python aoa_platform.py <base-url> <user> <password>`.

```
last  C3:10:00:00:00:02  Laptop QA-12     floor 1  x=11.00  y=1.62 m  at 2026-11-20T13:02:42.000Z
last  C3:10:00:00:00:03  Visitor badge 21 floor 1  x=11.60  y=6.00 m  at 2026-11-20T13:02:42.000Z
pos   C3:10:00:00:00:01  floor 1  x=5.17  y=4.38 m  ±1 m  zone=None
pos   C3:10:00:00:00:03  floor 1  x=10.80  y=6.00 m  ±1 m  zone=None
zone  C3:10:00:00:00:03  enter Corridor east  alert=True
```

Output (demo site)

Node.js: platform positions, snapshot + live

```
// Identium RTLS platform: AoA tag positions over REST (snapshot) and WebSocket (live).
// Node.js 18 or later.   npm install ws    then    node aoa-platform.js http://192.168.1.20:3004
<username> <password>
const WebSocket = require('ws');
const [BASE = 'http://localhost:3004', USER = 'api-reader', PASSWORD = ''] = process.argv.slice(2);
let cookie = '';

async function api(method, path, body) {
  const r = await fetch(BASE + path, { method, body: body && JSON.stringify(body),
    headers: { 'Content-Type': 'application/json', Cookie: cookie } });
  const set = r.headers.get('set-cookie');
  if (set) cookie = set.split(';')[0];              // keep the session cookie
  if (!r.ok) throw new Error(`${method} ${path}: HTTP ${r.status}`);
  return r.json();
}

async function start() {
  await api('POST', '/api/auth/login', { username: USER, password: PASSWORD });
  const { positions } = await api('GET', '/api/ble/positions');   // 1. snapshot: last known position
  for (const p of positions.filter(p => p.source === 'ble_aoa'))
    console.log(`last  ${p.beacon_mac}  ${p.beacon_name || '-'}  floor ${p.floor_id}
x=${p.x_meters.toFixed(2)}  y=${p.y_meters.toFixed(2)} m  at ${p.computed_at}`);

  const ws = new WebSocket(BASE.replace(/^http/, 'ws') + '/ws', { headers: { Cookie: cookie } });   //
2. live
  ws.on('message', (raw) => {
    const { type, data } = JSON.parse(raw);
    if (type === 'ble_positions' && data.source === 'ble_aoa')
      for (const p of data.positions)
        console.log(`pos   ${p.beacon_mac}  floor ${p.floor_id}  x=${p.x.toFixed(2)}
y=${p.y.toFixed(2)} m  ±${p.accuracy_meters} m`);
    else if (type === 'ble_zone_event')
      console.log(`zone  ${data.beacon_mac}  ${data.event_type}  ${data.zone_name}
alert=${data.alert}`);
    else if (type === 'ble_alert')
      console.log(`alert ${data.alert_type}: ${data.message}`);
  });
  ws.on('error', (e) => console.error('websocket:', e.message));
  ws.on('close', () => setTimeout(restart, 5000));   // sign in again and reconnect
}
const restart = () => start().catch((e) => { console.error(e.message); setTimeout(restart, 5000); });
restart();
```

kit/node/aoa-platform.js. Needs Node.js 18 or later: `npm install` in the kit's `node` folder (installs `ws`), then `node aoa-platform.js <base-url> <user> <password>`.

Python: standalone engine, UDP subscription

```
#!/usr/bin/env python3
# Standalone Identium Location Engine: subscribe to the UDP position feed.
# Python 3 standard library only.      python aoa_engine_udp.py <engine-ip> [port]
import json, socket, sys, time, zlib

HOST = sys.argv[1] if len(sys.argv) > 1 else '127.0.0.1'
PORT = int(sys.argv[2]) if len(sys.argv) > 2 else 55555

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.settimeout(1.0)
renewed = 0.0
while True:
    if time.time() - renewed > 30:                # a subscription must be renewed every 30 s
        sock.sendto(b'subscribe:zlib', (HOST, PORT))
        renewed = time.time()
    try:
        data, _ = sock.recvfrom(65535)
    except socket.timeout:
        continue
    if data[:1] not in (b'{', b'['):
        data = zlib.decompress(data, 47)          # zlib or gzip
    msg = json.loads(data)
    if msg.get('type') == 'sensors':              # tag positions, keyed by tag MAC
        for mac, t in msg['data'].items():
            print(f"pos   {mac}  map {t['mapId']}  x={t['x']:.2f}  y={t['y']:.2f}  z={t['z']:.2f} m"
                  f"  anchor {t.get('nearestGateway')}  at {t['calculatedAt']}")
    elif msg.get('type') == 'locators':           # anchors, keyed by anchor IP address
        for ip, a in msg['data'].items():
            print(f"anchor {a['mac']}  {ip}  online={a['online']}")
```

kit/python/aoa_engine_udp.py. Standard library only: `python aoa_engine_udp.py <engine-ip> 55555`.

Node.js: standalone engine, WebSocket

```
// Standalone Identium Location Engine: live positions over WebSocket.
// Node.js 18 or later.   npm install ws    then    node aoa-engine-ws.js ws://192.168.1.30:44444
const WebSocket = require('ws');
const zlib = require('zlib');
const URL = process.argv[2] || 'ws://127.0.0.1:44444';

function connect() {
  const ws = new WebSocket(URL);
  ws.on('open', () => console.log('connected to', URL));
  ws.on('message', (buf) => {
    const text = (buf[0] === 0x7b || buf[0] === 0x5b) ? buf.toString() :
zlib.unzipSync(buf).toString();  // JSON, zlib or gzip
    const msg = JSON.parse(text);
    if (msg.type === 'sensors')                       // tag positions, keyed by tag MAC
      for (const [mac, t] of Object.entries(msg.data))
        console.log(`pos   ${mac}  map ${t.mapId}  x=${t.x.toFixed(2)}  y=${t.y.toFixed(2)}
z=${t.z.toFixed(2)} m  anchor ${t.nearestGateway}`);
    else if (msg.type === 'locators')                 // anchors, keyed by anchor IP address
      for (const [ip, a] of Object.entries(msg.data))
        console.log(`anchor ${a.mac}  ${ip}  online=${a.online}  x=${a.info?.x} y=${a.info?.y}
z=${a.info?.z}`);
  });
  ws.on('error', (e) => console.error('websocket:', e.message));
  ws.on('close', () => setTimeout(connect, 3000));
}
connect();
```

kit/node/aoa-engine-ws.js: `node aoa-engine-ws.js ws://<engine-ip>:44444`. For a one-off read, `curl http://<engine-ip>:44444/beacons` returns the same tag objects as JSON.

11 / Configuration Reference

Floors and anchors are set up by the site's administrator in the platform (or through the API below). Engine settings live on the server and are set by Identium: tell us the values you need.

Floor (map)

Field	Meaning
`name`	Floor name
`width_meters`, `height_meters`	Real size of the plan area; sets the scale
`ceiling_m`	Ceiling height (metres)
`origin_bottom`	false (default): origin top-left, +Y down. true: origin bottom-left, +Y up
Floor-plan image	Uploaded on the Floor Plans page

Anchor

Field	Meaning
`locator_mac`	Anchor MAC from its label; the key (cannot be changed)
`name`, `model`	Display name; `ID-AOA-GW25`
`floor_id`, `x_meters`, `y_meters`	Floor, and the point on the plan below the anchor's centre
`z_meters`	Height of the anchor's face above the floor
`mount`, `azimuth`, `elevation`, `rotation`	`ceiling`; LED / port direction (0 = −Y, 90 = +X, 180 = +Y, 270 = −X); 0; 0
`ip_address`	Optional; filled in when the engine hears the anchor

Through the API: `POST /api/ble/locators` creates or updates an anchor by MAC (fields left out are kept), `PUT /api/ble/locators/{id}` changes only the fields sent, `DELETE /api/ble/locators/{id}` removes it. The AoA Anchors page also imports a CSV with the columns `mac, name, x, y, z, azimuth, elevation, rotation, floor`.

Engine and platform settings (server)

Setting	Default	What it does
Tag height `AOA_IQ_TAG_HEIGHT`	1.2 m	Height used for one-anchor fixes: about 0.75 m for desk and equipment tags, about 1.1 m for worn badges
Anchor data port `AOA_IQ_UDP_PORT`	55554	UDP port the anchors stream to
Publish interval `AOA_EMIT_INTERVAL_MS`	1000 ms	How often each tag's position is published
Tracking `AOA_TRACK`	on	Holds resting tags still and confirms moves; off = 4 s median smoothing
Freshness `AOA_TRACK_FRESH_MS`	15000 ms	A tag without a usable fix for this long stops being published
Offline alert `TAG_OFFLINE_MIN`	10 min	When a registered tag raises `tag_offline`
Webhook `ALERT_WEBHOOK_URL`, `ALERT_WEBHOOK_TOKEN`	off	Where alerts are posted, and the token sent with them

12/ Troubleshooting

Symptom	Likely cause	What to do
`401 Not authenticated`	Not signed in, session older than 24 h, or the server restarted	Sign in again and send the new cookie
WebSocket refused with 401	Connected without the session cookie	Pass the cookie in the `Cookie` header of the WebSocket request
No `ble_positions` with source `ble_aoa`	No AoA tag located	Check `GET /api/ble/aoa/status` (frames climbing, your anchors and tags listed); hold a tag face up below an anchor
Positions mirrored or rotated	Anchor azimuth or position entered wrongly	See *ID-AOA-GW25 User Manual*, section 10
y looks upside down	Floor frame	Check the floor's `origin_bottom` (section 03)
Tag drops out of the live feed	No usable fix for 15 s	Check the tag's interval (1 s or faster), orientation and coverage
Old positions in REST	Tag no longer located	Use `computed_at` to show the position as stale
B: no UDP messages	Subscription not renewed, another subscriber, or firewall	Send `subscribe:…` every 30 s; ask Identium to allow more subscribers or open the port
B: connection refused	Feed open only on the engine's own computer	Ask Identium to open it to your network
B: unreadable bytes	Compressed frame	Inflate with zlib or gzip before parsing
Webhook not arriving	Your URL not reachable from the server, or a non-2xx reply	Test the URL from the server's network; reply 200; check the token

13/ Support

For help with installation, integration or a fault that the steps above do not clear, contact Identium. Please keep the product model, the software version shown in the app and a short description (or screenshot) of the problem ready.

Company	Identium Tech Solutions Pvt. Ltd.
Address	Plot No. 5, First Floor, Raja Dhirsain Marg, Garhi, Sant Nagar, East of Kailash, New Delhi 110065, India
Phone	+91 97181 99516 · 011-4714 7839
WhatsApp	+91 82877 32647
Email	info@identium.in
Software and documents	www.identiumone.com/resourcehub/
Buy hardware and tags	www.indiarfidstore.com

This product and its software are updated from time to time; screens and specifications may change without notice. Check the Resource Hub for the latest version of this manual.