Automotus

CDS Events Integration Guide

This document describes the technical integration between **Automotus Events Notification Service (ENS)** and your organization's endpoint. It covers how Automotus will call your webhook, the exact payload format that will be delivered, and how to map Automotus zone IDs to physical locations.

---

1. Webhook Implementation

Overview

Automotus ENS will deliver CDS (Curb Data Specification) events to your HTTPS endpoint via HTTP POST. Your organization must expose a publicly accessible HTTPS endpoint that accepts JSON arrays of curb events.

Alternative integration: *If a webhook is not suitable for your infrastructure, ENS also supports delivery via* ***Azure Service Bus****. Contact the Automotus integration team via product@automotus.ai to discuss this option.*

Request Details

Property	Value
Method	`POST`
Content-Type	`application/json`
Body	JSON array of curb event objects
Expected success codes	`200`, `201`, `202`

Authentication

Your organization must provide Automotus with an **API key** that will be sent on every request in an HTTP header. You choose the header name (e.g., `X-API-KEY`).

Example request headers:

```
None

POST /automotus/events HTTP/1.1
Host: api.yourcompany.com
Content-Type: application/json
X-API-KEY: <your-api-key>
```

Your endpoint should validate the `X-API-KEY` header on every incoming request and reject requests with a missing or invalid key with a `401` or `403` status code.

Security note: *ENS does not include a request timestamp or signature beyond the API key. If replay attack protection is a concern for your implementation, it is your organization's responsibility to implement additional mitigations (e.g., timestamp validation using* `event_publication_time`*, deduplication by* `event_id`*).*

Retry Behavior

If the endpoint returns a non-success HTTP status code, ENS will retry the delivery with exponential backoff:

Attempt	Delay before retry
1st retry	1 second
2nd retry	2 seconds
3rd retry	4 seconds

After 3 failed attempts the event batch is dropped and logged internally.

Minimal Endpoint Implementation

Your endpoint must:

1. Accept `POST` requests with a JSON body
2. Validate the API key header
3. Parse the JSON array (see payload format below)
4. Return a `200 OK` (or `201`/`202`) promptly — **do not perform slow processing synchronously**; acknowledge receipt and process asynchronously
5. Return `4xx` or `5xx` on failure so ENS can retry

---

2. Event Configuration

At onboarding, your organization selects which event types to receive and configures data privacy settings. The available options are described below.

Available Event Types

Event Type	Description
`park_start`	A vehicle stopped or parked at a curb zone
`park_end`	A parked vehicle left the curb zone
`double_park_start`	A vehicle started double parking (blocking a travel lane or other parked vehicle)
`double_park_end`	A double-parked vehicle left

Partners can subscribe to any combination of the above. By default, all four event types are enabled.

License Plate Data

By default, **license plate data is not included** in payloads. If your organization requires license plate information (e.g., `vehicle_license_plate`), this must be explicitly requested and requires appropriate authorization. When enabled, the field is formatted as:

```
{2-char country code}_{2-char state code}_{plate}
(all uppercase)

Example: US_GA_ABC1234
```

---

3. Payload Format

ENS delivers a **JSON array** of curb event objects in a single POST call. Each object conforms to the CDS Curb Events specification.

Top-level envelope

The body is a plain JSON array — there is no outer wrapper object:

```
[
  { ...event... },
  { ...event... }
]
```

Batch sizes: *Each POST call may contain one or more events. Partners should expect arrays of varying sizes and handle them accordingly rather than assuming single-event payloads.*

Curb Event Object

Only fields that Automotus populates are listed below. Fields absent from this table are not sent.

Field	Type	Always present	Description
`event_id`	`string (UUID)`	Yes	Unique identifier for this event
`event_type`	`string`	Yes	One of: `"park_start"`, `"park_end"`, `"double_park_start"`, `"double_park_end"`
`event_session_id`	`string (UUID)`	When available	Groups a start and its corresponding end event together. Present when the sensor can correlate the two events; may be absent if the session could not be established (e.g., the vehicle was already parked when monitoring began). Do not treat this as a guaranteed field for session correlation.
`event_location`	`object`	Yes	GeoJSON Feature — location of the event (see below)
`event_time`	`integer`	Yes	Unix timestamp in **milliseconds** when the event occurred

`event_publication_time`	`integer`	Yes	Unix timestamp in **milliseconds** when ENS published the event
`curb_zone_id`	`string (UUID)`	Yes	Automotus zone ID where the event occurred (see Zone Mapping section)
`data_source_operator_id`	`string (UUID)`	Yes	ID of the operator that owns the sensor
`data_source_operator_name`	`string`	Yes	Always `"Automotus"`
`data_source_type`	`string`	Yes	Type of sensor (e.g., `"camera"`)
`data_source_device_id`	`string (UUID)`	Yes	ID of the specific sensor that captured the event
`data_source_device_name`	`string`	Yes	Human-readable name of the sensor
`data_source_manufacturer`	`string`	Yes	Sensor manufacturer
`data_source_model`	`string`	Yes	Sensor model
`sensor_status_is_online`	`boolean`	Yes	Always `true` — offline sensors do not generate events, so this field will never be `false` in practice
`vehicle_type`	`string`	When detected	See vehicle type values below

`vehicle_length`	`integer`	When detected	Estimated vehicle length in **centimeters**
`vehicle_propulsion_types`	`array<string>`	When detected	e.g., `["combustion"]`, `["electric"]`
`vehicle_license_plate`	`string`	When enabled	Only included if your organization has opted in and is authorized to receive license plate data. Format: `{country}_{state}_{plate}` in uppercase (e.g., `US_GA_ABC1234`)
`curb_occupants`	`array<object>`	Yes	Occupancy details within the zone (see below)

`event_location` Object

The location follows the GeoJSON Feature format. Coordinates are `[longitude, latitude]`.

None

```
{
  "type": "Feature",
  "properties": {
    "timestamp": 1712345678000
  },
  "geometry": {
    "type": "Point",
    "coordinates": [-84.40644495, 33.77471105]
  }
}
```

Field	Description
`type`	Always `"Feature"`
`properties.timestamp`	Unix timestamp in milliseconds — same value as `event_time`
`geometry.type`	Always `"Point"`
`geometry.coordinates`	`[longitude, latitude]` as decimal degrees

`curb_occupants` Array

Each element describes a vehicle occupying space within the zone:

None

```
[
  {
    "type": "car"
  }
]
```

Field	Type	Description
`type`	`string`	Type of occupant — one of the vehicle type values listed below (e.g., `"car"`)

Vehicle Type Values

Possible values for `vehicle_type` and `curb_occupants[].type`:

Value	Description
`car`	Passenger car or light-duty vehicle
`truck`	Light or heavy duty 4-wheeled truck
`van`	Van with significant interior cargo space
`motorcycle`	Seated mobility device for high-speed travel
`moped`	Seated fully-motorized low-speed device
`scooter`	Standing or seated fully-motorized device
`bicycle`	Two-wheeled personal mobility device
`cargo_bicycle`	Two- or three-wheeled cargo bicycle
`freight`	Large delivery truck with attached cab
`other`	Does not fit other categories
`unspecified`	Type not determined

---

4. Full Payload Example

The example below shows three events in a single POST call: a regular parking session (`park_start` + `park_end` linked by `event_session_id`) and a double-park event.

None


```
[
  {
    "event_id": "a1b2c3d4-0001-0001-0001-000000000001",
    "event_type": "park_start",
    "event_session_id":
"f9e8d7c6-0002-0002-0002-000000000002",
    "event_location": {
      "type": "Feature",
      "properties": {
        "timestamp": 1712345678000
      },
      "geometry": {
        "type": "Point",
        "coordinates": [-84.40644495, 33.77471105]
      }
    },
    "event_time": 1712345678000,
    "event_publication_time": 1712345680000,
    "curb_zone_id":
"11111111-aaaa-bbbb-cccc-000000000001",
    "data_source_operator_id":
"dddddddd-0000-0000-0000-000000000001",
    "data_source_operator_name": "Automotus",
    "data_source_type": "camera",
    "data_source_device_id":
"eeeeeeee-0000-0000-0000-000000000001",
    "data_source_device_name": "CAM-ATL-001",
```

```
    "data_source_manufacturer": "Axis",
    "data_source_model": "P3245-V",
    "sensor_status_is_online": true,
    "vehicle_type": "car",
    "vehicle_length": 450,
    "vehicle_propulsion_types": ["combustion"],
    "curb_occupants": [
      {
        "type": "car"
      }
    ]
  },
  {
    "event_id": "a1b2c3d4-0001-0001-0001-000000000003",
    "event_type": "park_end",
    "event_session_id":
"f9e8d7c6-0002-0002-0002-000000000002",
    "event_location": {
      "type": "Feature",
      "properties": {
        "timestamp": 1712346300000
      },
      "geometry": {
        "type": "Point",
        "coordinates": [-84.40644495, 33.77471105]
      }
    },
    "event_time": 1712346300000,
```

```
    "event_publication_time": 1712346302000,
    "curb_zone_id":
"11111111-aaaa-bbbb-cccc-000000000001",
    "data_source_operator_id":
"dddddddd-0000-0000-0000-000000000001",
    "data_source_operator_name": "Automotus",
    "data_source_type": "camera",
    "data_source_device_id":
"eeeeeeee-0000-0000-0000-000000000001",
    "data_source_device_name": "CAM-ATL-001",
    "data_source_manufacturer": "Axis",
    "data_source_model": "P3245-V",
    "sensor_status_is_online": true,
    "vehicle_type": "car",
    "vehicle_length": 450,
    "vehicle_propulsion_types": ["combustion"],
    "curb_occupants": []
  },
  {
    "event_id": "b2c3d4e5-0003-0003-0003-000000000004",
    "event_type": "double_park_start",
    "event_location": {
      "type": "Feature",
      "properties": {
        "timestamp": 1712346500000
      },
      "geometry": {
        "type": "Point",
```

```
        "coordinates": [-84.40517795, 33.77378005]
      }
    },
    "event_time": 1712346500000,
    "event_publication_time": 1712346502000,
    "curb_zone_id":
"22222222-aaaa-bbbb-cccc-000000000002",
    "data_source_operator_id":
"dddddddd-0000-0000-0000-000000000001",
    "data_source_operator_name": "Automotus",
    "data_source_type": "camera",
    "data_source_device_id":
"eeeeeeee-0000-0000-0000-000000000002",
    "data_source_device_name": "CAM-ATL-002",
    "data_source_manufacturer": "Axis",
    "data_source_model": "P3245-V",
    "sensor_status_is_online": true,
    "vehicle_type": "van",
    "vehicle_length": 550,
    "vehicle_propulsion_types": ["combustion"],
    "curb_occupants": [
      {
        "type": "van"
      }
    ]
  }
]
```

Events 1 and 2 share the same `event_session_id`, linking the `park_start` and `park_end` for the same parking session. Event 3 is an independent `double_park_start` with no session ID (session could not be established at time of publication).

---

5. Zone ID Mapping

The `curb_zone_id` field in every event refers to an Automotus-managed curb zone. Automotus will provide your organization with the full zone catalog at onboarding, and will notify you of any additions or changes.

Example Zone Mapping

curb_zone_id	City	Address	Latitude	Longitude	Zone Type	Length (m)
11111111-aaaa-bbbb-cccc-000000000001	City of Atlanta	818 Marietta St NW	33.77471	-84.40644	curb	14.45
22222222-aaaa-bbbb-cccc-000000000002	City of Atlanta	794 Marietta St NW	33.77378	-84.40518	curb	14.45

***Note:** The UUIDs above are illustrative. The actual zone IDs and their coordinates will be provided in the zone catalog delivered at onboarding. The zone catalog is not available as a real-time API — any additions or changes will be communicated directly by the Automotus integration team.*

Using Zone IDs

- Use `curb_zone_id` to correlate events to a physical location
- A start and end event with the same `event_session_id` occurred at the same zone

---

6. Questions

For questions about this integration, contact the Automotus integration team at **product@automotus.ai**.