API

- cuteTrac Vehicle Position API

cuteTrac Vehicle Position API

cuteTrac Vehicle Position API

Developer Integration Guide

API Name: Vehicle Position API
Purpose: Retrieve the latest GPS position and vehicle status information for vehicles registered in the cuteTrac fleet management system.

---

1. API Endpoint

Production Endpoint

```
GET https://www.cutetrac.com/api/cuteTracAPIServicePUB.svc/position
```

Request URL

```
https://www.cutetrac.com/api/cuteTracAPIServicePUB.svc/position?key={API_KEY}&fmt=json
```

HTTP Method

```
GET
```

Response Format

```
JSON
```

2. Authentication

The API requires an API key to authenticate the request.

Query Parameters

Parameter	Required	Description	Example
`key`	Yes	API authentication key issued by cuteTrac	`ABC123XYZ`
`fmt`	Yes	Response format	`json`

Example Request

```
GET https://www.cutetrac.com/api/cuteTracAPIServicePUB.svc/position?key=YOUR_API_KEY&fmt=json
```

> **Security:** The API key should be treated as confidential. Do not expose the API key in publicly accessible client-side JavaScript, mobile applications, or source code repositories. Where possible, make API calls through your application's backend/server.

3. Response

The API returns a JSON array containing the latest position information for vehicles.

Example Response


```
[
  {
    "AddressLocation": "187 Pandan Loop",
    "Asset": "YR 4548 U (Raja)",
    "AssetID": "163",
```

```
    "Battery": "0",
    "Course": "254.00",
    "DriverName": "",
    "DriverTel": "",
    "Fix": "GPS Fix",
    "FixID": "2",
    "Fuel": "0",
    "HDOP": "0",
    "Ignition": "0",
    "Mileage": "152253.11251000",
    "PosID": null,
    "PosX": "103.7536033000",
    "PosY": "1.3118916000",
    "PosZ": "0",
    "RxTime": "2026-09-09T13:57:28.000",
    "Satellites": "14",
    "Speed": "0.00",
    "Timestamp": "2026-09-09T13:57:28.000",
    "VehicleGrp": ""
  }
]
```


Each object in the returned array represents one vehicle/asset.

---

4. Response Fields

Field	Data Type	Description
`AddressLocation`	String	Approximate/current address or location of the vehicle
`Asset`	String	Vehicle registration number and associated driver/name, where configured
`AssetID`	String	Unique cuteTrac asset/vehicle ID
`Battery`	String/Number	Battery-related value reported by the tracking device
`Course`	String/Number	Direction of travel in degrees
`DriverName`	String	Driver assigned to the vehicle

Field	Data Type	Description
`DriverTel`	String	Driver telephone number, if configured
`Fix`	String	GPS fix status
`FixID`	String/Number	Numeric GPS fix status identifier
`Fuel`	String/Number	Fuel value reported by the vehicle/device, where supported
`HDOP`	String/Number	Horizontal Dilution of Precision (GPS accuracy indicator)
`Ignition`	String/Number	Vehicle ignition status
`Mileage`	String/Number	Vehicle accumulated mileage
`PosID`	String/Number/Null	Position record identifier, if available
`PosX`	String/Number	Longitude
`PosY`	String/Number	Latitude
`PosZ`	String/Number	Altitude/elevation
`RxTime`	DateTime	Time the position data was received by the system
`Satellites`	String/Number	Number of GPS satellites detected
`Speed`	String/Number	Current vehicle speed
`Timestamp`	DateTime	Timestamp associated with the GPS position
`VehicleGrp`	String	Vehicle group assigned to the asset

---

5. Important GPS Fields

The following fields are particularly useful when integrating cuteTrac with another application.

Latitude

```
PosY
```

Example:

```
"PosY": "1.3118916000"
```

Longitude

```
PosX
```

Example:

```
"PosX": "103.7536033000"
```

Therefore, the GPS coordinate is:

```
Latitude  = 1.3118916000
Longitude = 103.7536033000
```

These coordinates can be used with mapping platforms such as Google Maps, OpenStreetMap, or other mapping APIs.

---

6. Vehicle Speed

The `Speed` field represents the current vehicle speed.

Example:

```
"Speed": "0.00"
```

A value of:

```
0.00
```

indicates that the vehicle is currently stationary according to the latest GPS position.

> **Note:** The unit of speed should be confirmed according to the specific cuteTrac account/device configuration before displaying or converting the value in another application.

---

7. Vehicle Ignition

The `Ignition` field indicates the ignition status reported by the tracking device.

Example:

```
"Ignition": "0"
```

The application developer should use the agreed cuteTrac ignition status mapping when displaying this information.

For example, depending on the device configuration:

```
0 = OFF
1 = ON
```

The exact mapping should be confirmed with cuteTrac before implementing business logic based on this field.

---

8. Direction / Course

The `Course` field represents the vehicle's heading in degrees.

Example:

```
"Course": "254.00"
```

The value is measured clockwise from North:

```
0°   = North
90°  = East
180° = South
270° = West
```

This value can be used to rotate a vehicle icon on a map to indicate the vehicle's direction of travel.

---

9. GPS Fix

The API provides GPS fix information through:

```
Fix
FixID
```

Example:

```
"Fix": "GPS Fix",
"FixID": "2"
```

`Fix` provides a descriptive status, while `FixID` provides the corresponding numeric identifier.

Applications should preferably use the `Fix` value for displaying the status and use `FixID` only when implementing status-based business logic.

---

10. Timestamp

The API provides two timestamp-related fields:

```
RxTime
Timestamp
```

Example:

```
"RxTime": "2026-09-09T13:57:28.000",
"Timestamp": "2026-09-09T13:57:28.000"
```

The values use the following general format:

```
YYYY-MM-DDTHH:mm:ss.fff
```

Example:

```
2026-09-09T13:57:28.000
```

Developers should parse these values as DateTime values rather than treating them as ordinary display strings.

---

11. Mileage

The `Mileage` field contains the accumulated mileage reported for the vehicle.

Example:

```
"Mileage": "152253.11251000"
```

Applications should format the value appropriately for display.

For example:

```
152,253.11
```

> The unit of mileage should be confirmed according to the cuteTrac configuration before displaying the value as kilometres or miles.

---

12. Driver Information

The API can return driver information associated with the vehicle.

Example:

```
"DriverName": "Raja",
"DriverTel": "..."
```

However, these fields may be empty:

```
"DriverName": "",
"DriverTel": ""
```

Applications must therefore handle empty strings and should not assume driver information is always available.

---

13. Address Information

The `AddressLocation` field provides the current/last known address associated with the GPS position.

Example:

```
"AddressLocation": "187 Pandan Loop"
```

The address can be displayed alongside the vehicle's location on a dashboard or vehicle tracking interface.

---

14. Multiple Vehicles

The API returns an array because multiple vehicles may be returned in a single request.

Example:

```
[
  {
    "Asset": "YR 4548 U (Raja)",
    "AssetID": "163",
    "PosX": "103.7536033000",
    "PosY": "1.3118916000",
    "Speed": "0.00"
  },
  {
    "Asset": "YR 6682 B (Ganesan)",
    "AssetID": "1706",
    "PosX": "103.7108416000",
    "PosY": "1.3351050000",
    "Speed": "0.00"
  }
```

```
]
```

The consuming application should iterate through the returned array and process each vehicle independently.

---

15. Example Integration Flow

A typical integration can follow this process:

```
1. Obtain API Key from cuteTrac
          |
          v
2. Send GET request to Position API
          |
          v
3. Receive JSON response
          |
          v
4. Parse JSON array
          |
          v
5. Process each vehicle
          |
          +--> Vehicle ID
          +--> Vehicle Registration
          +--> Driver
          +--> Latitude / Longitude
          +--> Speed
          +--> Ignition
          +--> Direction
          +--> Mileage
          +--> GPS Status
          |
          v
6. Display/store information in customer's application
```

16. Example JavaScript Request

```
const apiKey = "YOUR_API_KEY";

const url =
  `https://www.cutetrac.com/api/cuteTracAPIServicePUB.svc/position` +
  `?key=${encodeURIComponent(apiKey)}&fmt=json`;

fetch(url)
  .then(response => {
    if (!response.ok) {
      throw new Error(`HTTP Error: ${response.status}`);
    }

    return response.json();
  })
  .then(data => {
    data.forEach(vehicle => {
      console.log("Vehicle:", vehicle.Asset);
      console.log("Asset ID:", vehicle.AssetID);
      console.log("Latitude:", vehicle.PosY);
      console.log("Longitude:", vehicle.PosX);
      console.log("Speed:", vehicle.Speed);
      console.log("Ignition:", vehicle.Ignition);
      console.log("Timestamp:", vehicle.Timestamp);
    });
  })
  .catch(error => {
    console.error("cuteTrac API Error:", error);
  });
```

17. Example C# Integration

```
using System.Net.Http;
using System.Text.Json;

public async Task GetVehiclePositions(string apiKey)
{
    using var client = new HttpClient();

    string url =
        $"https://www.cutetrac.com/api/cuteTracAPIServicePUB.svc/position" +
        $"?key={Uri.EscapeDataString(apiKey)}&fmt=json";

    var response = await client.GetAsync(url);

    response.EnsureSuccessStatusCode();

    string json = await response.Content.ReadAsStringAsync();

    var vehicles = JsonSerializer.Deserialize<List<VehiclePosition>>(json);

    foreach (var vehicle in vehicles)
    {
        Console.WriteLine($"Vehicle: {vehicle.Asset}");
        Console.WriteLine($"Latitude: {vehicle.PosY}");
        Console.WriteLine($"Longitude: {vehicle.PosX}");
        Console.WriteLine($"Speed: {vehicle.Speed}");
    }
}
```

Example model:

```
public class VehiclePosition
{
    public string AddressLocation { get; set; }
    public string Asset { get; set; }
    public string AssetID { get; set; }
    public string Battery { get; set; }
```

```
    public string Course { get; set; }
    public string DriverName { get; set; }
    public string DriverTel { get; set; }
    public string Fix { get; set; }
    public string FixID { get; set; }
    public string Fuel { get; set; }
    public string HDOP { get; set; }
    public string Ignition { get; set; }
    public string Mileage { get; set; }
    public string PosID { get; set; }
    public string PosX { get; set; }
    public string PosY { get; set; }
    public string PosZ { get; set; }
    public string RxTime { get; set; }
    public string Satellites { get; set; }
    public string Speed { get; set; }
    public string Timestamp { get; set; }
    public string VehicleGrp { get; set; }
}
```

---

18. Mapping Integration Example

For a mapping application:

```
PosY = Latitude
PosX = Longitude
Course = Vehicle Heading
Speed = Vehicle Speed
Asset = Vehicle Display Name
AddressLocation = Current Address
```

For example:

```
Vehicle:
YR 4548 U (Raja)

Location:
Latitude:   1.3118916
Longitude:  103.7536033

Speed:
0.00

Direction:
254°

Address:
187 Pandan Loop
```

The consuming application can use `PosX` and `PosY` to place a vehicle marker on a digital map.

---

19. Handling Null and Empty Values

Developers must handle fields that may contain:

```
null
""
"0"
```

For example:

```
"PosID": null,
"DriverName": "",
"DriverTel": "",
"VehicleGrp": ""
```

The application should not assume that every field contains a value.

Recommended handling:

```
null       → Treat as unavailable
""         → Treat as unavailable/not configured
"0"        → Treat according to the specific field definition
```

---

20. Recommended Polling

The Position API provides the latest available vehicle position.

If the customer's application requires near-real-time tracking, the application may periodically call the API.

For example:

```
Every 10 seconds
Every 30 seconds
Every 60 seconds
```

The polling interval should be selected based on the customer's requirements and agreed API usage limits.

> Developers should avoid excessive polling. The recommended polling frequency and any API request limits should be confirmed with cuteTrac before deployment.

---

21. Error Handling

The consuming application should handle common HTTP/API failures such as:

```
HTTP 400 – Invalid Request
HTTP 401/403 – Authentication/Authorization Failure
HTTP 404 – Endpoint Not Found
HTTP 429 – Too Many Requests
HTTP 500 – Server Error
```

```
HTTP 503 – Service Unavailable
```

The application should implement appropriate retry and error-handling mechanisms for temporary service failures.

---

22. API Key Security

The API key grants access to cuteTrac data and must be protected.

Recommended

```
Customer Application
        |
        v
Customer Backend Server
        |
        | API Key
        v
cuteTrac API
```

Avoid

```
Browser / Mobile App
        |
        | API Key exposed
        v
cuteTrac API
```

The API key should preferably be stored in a secure server-side configuration or secrets-management system.

---

23. Field Summary

Category	Fields
Vehicle	`Asset`, `AssetID`, `VehicleGrp`
Driver	`DriverName`, `DriverTel`
Location	`AddressLocation`, `PosX`, `PosY`, `PosZ`
Movement	`Speed`, `Course`
Vehicle Status	`Ignition`, `Battery`, `Fuel`
GPS	`Fix`, `FixID`, `HDOP`, `Satellites`
Distance	`Mileage`
Time	`RxTime`, `Timestamp`
Position Reference	`PosID`

---

24. Contact / API Access

For API access, API key provisioning, field interpretation, request limits, or integration assistance, please contact the cuteTrac technical support team.

Platform: cuteTrac Fleet Management System
API: Vehicle Position API
Endpoint: `https://www.cutetrac.com/api/cuteTracAPIServicePUB.svc/position`

Document Notes

This document describes the current response structure and provides guidance for third-party developers integrating with cuteTrac.

Specific field mappings such as **ignition status, fuel units, mileage units, speed units, API request limits, and API key permissions** should be confirmed with cuteTrac before production implementation.