Programmatic Access to the Azuga ELD Platform

Retrieving and manipulating the key structures in the Azuga ELD system through a set of RESTful APIs

Version 5

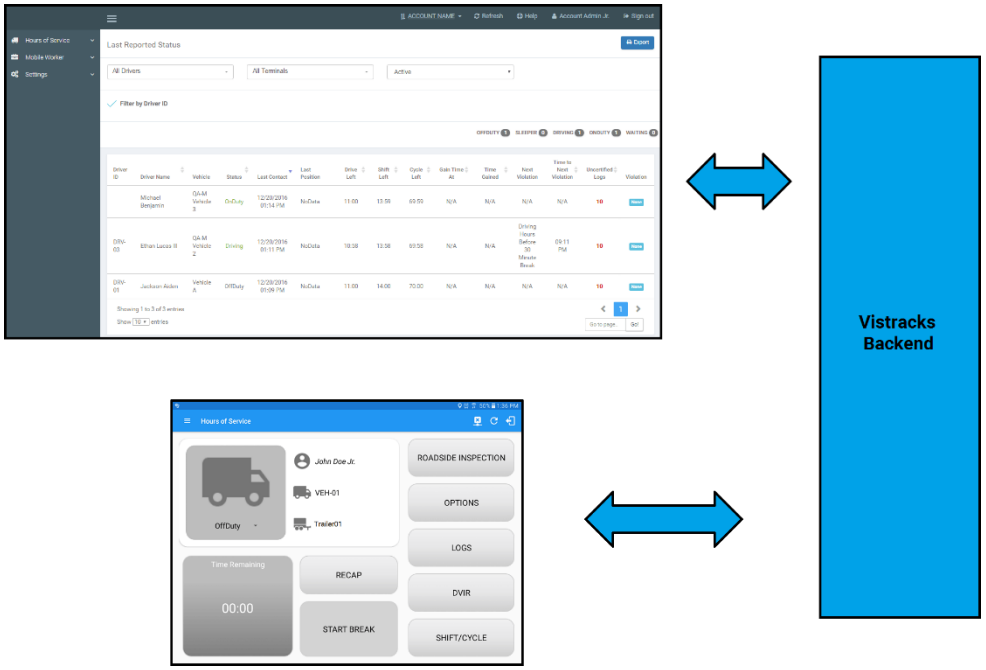
Azuga ELD
March 04, 2024

Contents

Introduction

Azuga ELD provides a powerful and comprehensive solution for managing the movements and activities of mobile workers in general, and drivers subject to federal "hours of service" regulations in particular. The full solution comprises a front-end app typically running on a tablet form-factor device, an administrative portal providing access to information about a fleet of vehicles, and a "back-end" cloud-resident platform that monitors and manages the real-time information transmitted from the tablets and other telematics devices.

This document focuses specifically on the services that are offered by the back-end platform. These services, while used to support the tablet and portal interfaces provided by Azuga ELD, were designed to be equally accessible by other clients through a formal "RESTful" interface.[1] Details of individual REST calls are documented on the "developer.eldazuga.com" site.



Azuga ELD backend supports both mobile and platform clients through REST API

[1] See http://en.wikipedia.org/wiki/Representational_state_transfer for an overview of REST concepts

Prerequisites

Before Starting

Before attempting to work with the Azuga ELD APIs, it's necessary to have obtained an account on one of the Azuga ELD servers - typically either:

https://eld.azuga.com

or

https://staging.eld.azuga.com

You should have a username and password for an administrative user on that account. Ideally, the account should be set up with one or more devices and assets. These can be added from the main administration page after logging in to the appropriate server.

Accessing the Azuga ELD REST APIs

Authenticating REST requests

Azuga ELD provides a uniform method for manipulating the key objects that make up the back-end platform. As with many REST models, *objects*, identified by a particular URL, can be *created, read, updated,* or *deleted* through the standard HTTP operations of POST, GET, PUT, and DELETE respectively. The underlying state of an object is encoded as a JSON structure[2]. Parameters that may influence the request (for example, specifying a time range of interest in retrieving a set of records) are encoded as part of the query string associated with the URL. Finally, to access individual objects (rather than a full collection), it is generally possible to augment the base URL with the unique "id" of the object in question. For example,

https://eld.azuga.com/api/v2/assets

would return all assets in the account, while:

https://eld.azuga.com/api/v2/assets/143525

would return a single element from the collection.

Because Azuga ELD provides a "multi-tenant" architecture (meaning a number of distinct accounts are managed by a single server instance,) every API request must include some form of account authentication. The authentication consists of a username, password pair that is encoded using a mechanism known as "basic authentication". Details are described in the inset below.

[2] At present, all the APIs discussed in this document manipulate data in a JSON format. However, some of the Legacy APIs which use the XML format can still be accessed as part of the transition period. Plans going forward include the deprecation or removal of all old API endpoints and all API requests and responses should only use JSON format.

1. Obtain a valid username and password for your account
2. Encode the username and password:
   Using Base64 Encoding, encode the username and password with a colon separating them to obtain the *encoded credentials*
   e.g. `johnsmith@mail.com:mysecurepassword`
   `-> am9obnNtaXRoQG1haWwuY29tOm15c2VjdXJlcGFzc3dvcmQ=`
3. Put an HTTP request header called "Authorization" on any API request with the value of "Basic <*encoded credentials*>"
   e.g. `Authorization:`
   `Basic am9obnNtaXRoQG1haWwuY29tOm15c2VjdXJlcGFzc3dvcmQ=`
4. The HTTP response will include a "Set-Cookie" header that contains a short lived session id that can be used for additional authorization requests
   e.g. `Set-Cookie:`
   `JSESSIONID=4664C731A74FA93252FE9896FF389E2D; Path=/`
5. Use the JSESSIONID cookie on all subsequent API requests
   e.g. `Cookie: JSESSIONID=4664C731A74FA93252FE9896FF389E2D`

Authenticating an API request

There are some subtleties associated with this authentication scheme. If you're creating a browser-based application in JavaScript that will communicate with the Azuga ELD server using these REST APIs, it may not be possible to retrieve the JSESSIONID cookie as described in step 5. This is due to a security limitation imposed on applications in the browser environment. In this case, it's possible to simply include the encoded username/password pair with every API request.

As explained below, it is also possible to log in to the " eld.azuga.com" site in a separate browser tab and issue any API calls from another tab in the same browser. In this arrangement, the authenticated JSESSIONID is automatically sent to the Azuga ELD server with each API request without needing to explicitly add any headers.

Example API Request

Using a browser to explore an API

Note: in many instances it is possible to explore the available API set without writing any code. This facility relies on the ability present in most browsers to work in multiple tabs. Logging into the Azuga ELD portal (e.g., eld.azuga.com) with valid credentials establishes a persistent session identifier within the browser that will be used by API references from other tabs. Following the steps below should display the result of a simple API call to return information about all the users in an account.

1. Enter your username and password to log in. (You should see the main landing page)
2. Open a new tab in the same browser
3. From this new tab, enter the URL for an API, e.g.,

https://eld.azuga.com/api/v2/users

This should display an JSON structure similar to the following that enumerates the users associated with the account.

JSON returned by a "GET" request


```
▼ [
  ▶ { … }, // 13 items
  ▼ {
      "firstName": "John",
      "lastName": "Smith",
      "suffix": "Jr.",
      "email": "johnsmith@mail.com",
      "eulaAcceptedDate": "2017-03-06T23:52:58.570Z",
    ▼ "roles": [
      ▼ {
            "name": "ROLE_ACCOUNTADMIN"
        }
      ],
    ▼ "userRoles": [
          "USER_ROLE_ACCOUNTADMIN"
      ],
    ▶ "permissions": [ … ], // 19 items
    ▼ "visibilitySetIds": [
          2673
      ],
      "active": true,
      "accountId": 1500001,
      "id": 1500009,
      "lastChangedDate": "2017-03-09T07:05:12.591Z"
  },
  ▼ {
      "firstName": "Mike",
      "lastName": "Brown",
      "email": "mikebrown@mail.com",
      "eulaAcceptedDate": "2017-03-07T01:34:13.209Z",
    ▼ "roles": [
```


You can try accessing other objects, e.g., *assets*, in a similar fashion. (https://eld.azuga.com/api/v2/assets)

Use Query Parameters to access individual objects or objects matching a pattern.

This simple technique works because the browser issues an HTTP GET request to the supplied URL. The Azuga ELD system receives this request and, based on the associated authentication information, returns the corresponding data encoded as JSON. Normally, it would be the responsibility of the external programming making the API requests to format and issue this GET request.

To perform requests other than GET, (e.g., creating (POST) or modifying (PUT) an instance of an object) it would be necessary to write a program to properly format the request, or to use a debugging proxy tool like "Fiddler".[3]

When accessing a REST endpoint (e.g., the User object in the example above) the information returned often represents an unqualified collection of the objects. Individual objects, or subsets, can be specified either by appending a list of object-identifiers (separated by a '+' symbol) as part of the main URL, or by adding specific query parameters.

[3] See for example, http://www.telerik.com/fiddler

Example:

https://eld.azuga.com/api/v2/users	returns information about all users
https://eld.azuga.com/api/v2/users/1246844	returns information about a single user with the supplied id.
https://eld.azuga.com/api/v2/users/1246844+1246849	returns information about two users
https://eld.azuga.com/api/v2/users?first-name=John	returns information about all users with the given first name "John".

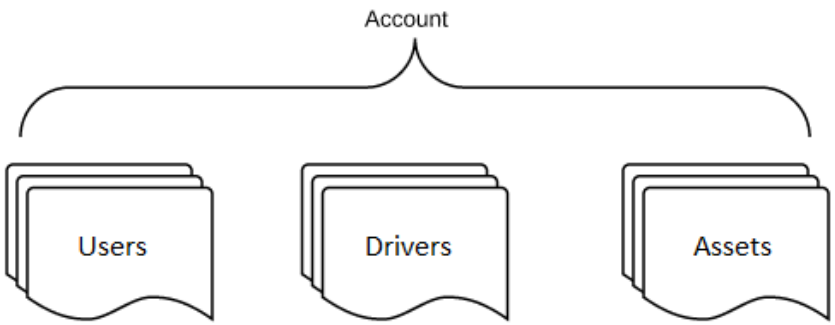
Fundamental Objects

In order to effectively use and manipulate the objects exposed by the Azuga ELD REST API, it is important to have some familiarity with the system's underlying data model.

This section will focus on the interrelationships among four key structures:

- Accounts
- Users
- Drivers
- Assets

All data objects accessible through the API are contained by a particular account.



All major data objects accessible to applications are contained under the umbrella of a particular account. In this section, we'll look at each of the three objects shown in the diagram above and discuss the ways in which they're inter-related.

Users

Each *user* is associated with a particular account and is identified by a globally unique email

One or more *Users* can be associated with a given account, each of whom is unique. In the example API request, each user is also given a unique numeric identifier generated by the system. (In the illustration above, this identifier is marked by the "id" field and has the value 1500009.) This numeric identifier will be important in several

cases as it provides a means of linking related records. For example, in Hours of Service (HOS) applications, one record type returns information about a driver's daily activity. The driver in question is represented within that record by an element such as

```
userId: 1500009
```

where the value of the *userId* element is the unique numeric id of a particular user.

Each user can also be assigned a set of *user roles.* A user's user roles determine what capabilities they have within the system. In the above JSON example, the user was designated as having USER_ROLE_ACCOUNTADMIN. The full list of default user roles and their access levels can be found on http://developer.eldazuga.com/index.php/user-roles-1704

Assets

Assets provide an abstraction to represent the difference between the object being monitored and the instrument doing the monitoring. For example, an *asset* might be a particular car in a fleet and may have attributes like its Vehicle Identification Number. The vehicle may be equipped with a telematics unit from a given manufacturer. The collected data, though, is conceptually associated with the asset and from an API perspective will be retrieved by referencing the *asset* object.

In the standard Azuga ELD clients (Azuga ELD web portal or Android/iOS app) the "Asset Name" is typically used to identify a given piece of equipment. Neither the name given to the supporting device, nor its unique identifier are usually visible to the end user.

Drivers

Drivers are a particular type of User that are allowed to access the Azuga ELD platform through the Hours of Service application in a smartphone or a tablet and perform Driving Events. Driving events are created by drivers operating an equipment. Any update using the 'api/v2/drivers' endpoint also reflects on the similar fields that can be found on the 'api/v2/users' endpoint, vice versa.

Drivers and Hours of Service Objects

The preceding material discussed some of the basic data constructs underlying the Azuga ELD system. The concepts of users, drivers, and assets pertain to a wide variety of vertical applications. This section focuses on objects that support a particular domain, that of mobile workers or professional drivers that will use some form of tablet or phone device to record their travels and duty status, inform them of new assignments, and support them in conducting and documenting any required inspections.

Many professional drivers are subject to a set of regulations defined by the "Federal Motor Carrier Safety Administration" that define specific guidelines for the amount of

time a driver can operate a vehicle without some form of break. The Hours of Service app offered by Azuga ELD assists a driver in complying with these regulations by allowing the driver to note when they transition among defined states (e.g., Driving, Off Duty, etc.) and presenting the driver with indications that they may be about to exceed a particular threshold. The accumulating information is also relayed by the app back to the Azuga ELD server where it can be accessed through additional API calls.

Six objects, listed in the following table, provide the bulk of the information relevant to understanding a particular driver's activity.

Object	Description
Driver	Summary information about a user registered within the system that will be driving a commercial vehicle subject to the FMSCA regulations.
Driver Status	A structure that aggregates all current information about a driver such as their current duty status, violations, etc.
Driver Daily	Summary information about an individual's activity on a particular day
Driver History	Records reflecting all status changes for a driver
Driver Violations	Information about driving violations incurred by a driver
DVIRS	Details of the assets that were inspected

Table 1 - APIs related to Hours of Service application

Pictorially, the relationships among various items is shown below:

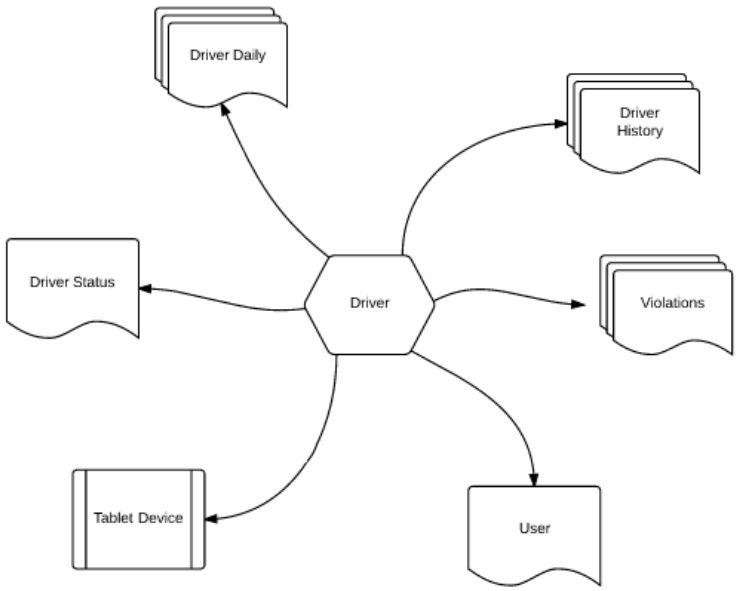


Image 1 - Relationships between a Driver object and associated data

In addition, two other objects, *Work Orders* and *Job Sites* provide information for the activities of mobile workers that may or may not be subject to the FMCSA regulations.

Driver Daily

The *Driver Daily* object returns information about a full day's activity for one or more drivers over a given time range. An example URL for this API includes a specific time range and an indication of whether records for all drivers are of interest:

https://eld.azuga.com/api/v2/driverDailies?all-users=true&from-log-date=2017-03-08&to-log-date=2017-03-08

In this example, there are several arguments included as part of the query string in the URL:

all-users=true indicated that the request should return information about all drivers in the account. To select only a specific driver, the userId for the individual in question can be added to the query string in place of the allUsers:

https://eld.azuga.com/api/v2/driverDailies?user-id=1500098&from-log-date=2017-03-08&to-log-date=2017-03-08

Alternatively, if the credentials passed with the request are for an individual user instead of a user with the role ACCOUNT_ADMIN, only records pertaining to that user will be returned.

from-log-date, to-log-date define the time range of interest.

As with all the API calls, the information returned from this takes the form of an JSON structure. The (somewhat lengthy) result of the example above appears as follows:

```
[
  {
    "userId": 1500098,
    "carrier": "",
    "carrierUsDotNumber": "",
    "certified": false,
    "coDriverName": "Mike Brown",
    "cycleUsa": "US70hr8days",
    "cycleCa": "Can70hr7daysSouth",
    "logDate": "2017-03-08",
    "driverEmail": "johndoe@mail.com",
    "driverName": "John Doe Jr.",
    "driverPhone": "",
    "timeZone": "Asia/Manila",
    "homeTerminalAddress": "",
    "mainOfficeAddress": "",
    "beginOdometer": 0,
    "endOdometer": 0,
    "shippingDocsManifestNo": "",
    "shippingDocsShipperCommodity": "",
    "startTimeOfDay": "00:00:00",
    "trailerId": "[\"Trailer01\",\"None\"]",
    "vehicleId": "[\"VEH-01\"]",
    "cargo": "PROPERTY",
    "fields": [],
    "driverDailyDocuments": [],
    "username": "johndoe@mail.com",
    "manualLog": false,
    "date": "2017-03-08",
    "accountId": 1500001,
    "id": 347710,
    "lastChangedDate": "2017-03-08T07:46:59.142Z"
  },
```


Principal fields in this structure are listed in the following table.

Object	Description
Cargo	Type of material being shipped as supplied on “Driver Settings” page of app
Carrier	Name of Carrier as supplied on “Driver Settings” page of app
Certified	Boolean indicating whether the day’s log had been certified by the driver
coDriverName	Optional second driver on duty
logDate	Date of Report
driverEmail	Email of *user* object associated with person logged into tablet
driverName	Name of user object associated with person logged into tablet

beginOdometer	Value entered by driver at start of day
endOdometer	Value entered by driver at end of day
startTimeOfDay	Time when the day of the driver starts
vehicleId	Identifier of truck entered in driver settings
trailerId	Identifier of trailer entered in driver settings
userId	Identifier of user of driver daily
manualLog	Boolean value indicating if the log was manually entered.

Table 2 – Principal fields returned from Driver Daily request

Driver History

Drivers using the Azuga ELD Hours of Service app are able to use the main interface screen to easily change their driving status. There are five defined states for a driver:

- Driving
- On Duty
- Off Duty
- Sleeper Berth
- Waiting at Site (for Oil Field Service Exception)

When the driver makes an explicit state change, they are prompted to provide an optional note documenting the change. In addition, the software will record the location of the device at the time the change was made. Aside from the five defined driving events, there are also secondary events that are recorded at different instances by the application.

The *Driver History* object provides access to these changes. Typically, the call to retrieve Driver History records would provide a date range of interest. The resulting JSON appears as follows:

```
{
    "userId": 1500098,
    "uuid": "45857a97-8787-4413-8042-9fa5a7ca1dbc",
    "location": "",
    "driverEdit": false,
    "note": "",
    "latitude": 1000,
    "longitude": 1000,
    "odometerKm": 0,
    "editReasonCode": "No reason selected",
    "useCycleReset": true,
    "validBeginTime": "2017-03-08T04:38:20.115Z",
    "transitionFromId": 5400958,
    "vin": "-VEH-0000000000001",
    "certificationCount": 0,
    "recordStatus": "Active",
    "engineHours": "PT0S",
    "username": "johndoe@mail.com",
    "distanceLastGpsKm": 0,
    "malfunctionIndicator": false,
    "diagnosticIndicator": false,
    "dataCheck": 28,
    "eventType": "Driving",
    "recordOrigin": "Driver",
    "eventTime": "2017-03-08T04:38:18.637Z",
    "assetId": 1354380,
    "eventSequenceIdentifier": 9,
    "accountId": 1500001,
    "id": 5400990,
    "lastChangedDate": "2017-03-08T04:29:49.302Z"
},
```


The key fields for each record are summarized in the table below:

Field	Description
userId	reference to the relevant driver whose status change is being reported
location	city and state where status change was initiated
driverEdit	True if driver edited this status record
note	optional text field containing any notes provided by the driver relating to the state change. These notes are reflected in the visual chart in the app and reports
validBeginTime	date and time stamp recording time when state change was initiated
vin	Vehicle Identification Number of the asset used by the driver when the event was recorded
recordStatus	The current status of the driver history

eventType	The new driving state – one of OffDuty, OnDuty, Sleeper, Driving, WaitingAtSite (or secondary event types)
recordOrigin	The origin of the record
eventTime	Date and time of the status change
assetId	Identifier of the asset used by the driver when the event was recorded
lastChangedDate	Time at which the most recent state change for the driver was recorded

Table 3 – key fields returned by the Driver History API

Driver Violations

The *Driver Violations* API includes a list of any violations incurred over a specified time range. Possible violations include:

Violation Type
CYCLE_DUTY_HOURS
SHIFT_DRIVE_HOURS
BREAK_DRIVE_HOURS
SHIFT_ELAPSED_HOURS
CAN_DAILY_DRIVE_HOURS
CAN_DAILY_DUTY_HOURS
CAN_SHIFT_DRIVE_HOURS
CAN_SHIFT_DUTY_HOURS
CAN_SHIFT_ELAPSED_HOURS
CAN_DAILY_OFF_DUTY_HOURS
CAN_24MAN_OFF_DUTY_HOURS
CAN_CYCLE1_DUTY_HOURS
CAN_CYCLE2_DUTY_HOURS
CAN_CYCLE2_24OFF_DUTY_HOURS
CAN_OIL_WELL_SERVICE

Table 4 - Possible Values of Driver Violations

Each violation includes a timestamp reflecting the date and time at which the violation occurred and in indication of the type of violation. Violations are returned in a JSON collection as illustrated below. Not only that each violation has its own unique id, but all refers to the userId of the associated driver that has incurred the violation.

```
[
  {
    "userId": 1250894,
    "driverHistoryId": 5105777,
    "timestamp": "2016-10-04T16:25:19.274Z",
    "violationName": "BREAK_DRIVE_HOURS",
    "iconLabel": "30",
    "message": "You have exceeded your driving limit of 8 hours before taking a 30 minute break",
    "title": "Driving Hours Before 30 Minute Break",
    "toolTipText": "30 Minute Rest Break Violation at 5-Oct-16 00:25",
    "accountId": 1248281,
    "id": 2139441,
    "lastChangedDate": "2016-10-05T08:02:14.482Z"
  },
  {
    "userId": 1250894,
    "driverHistoryId": 5105777,
    "timestamp": "2016-10-04T19:25:19.274Z",
    "violationName": "SHIFT_DRIVE_HOURS",
    "iconLabel": "11",
    "message": "You have exceeded your shift driving time of 11 hours",
    "title": "Driving Hours In Shift",
    "toolTipText": "11 Shift Driving Hour Violation at 5-Oct-16 03:25",
    "accountId": 1248281,
    "id": 2139442,
    "lastChangedDate": "2016-10-05T08:02:14.500Z"
  },
  {
    "userId": 1250894,
    "driverHistoryId": 5105777,
    "timestamp": "2016-10-04T22:25:19.274Z",
    "violationName": "SHIFT_ELAPSED_HOURS",
    "iconLabel": "14",
    "message": "You have exceeded your shift duty span limit of 14 hours",
    "title": "Duty Span Hours In Shift",
    "toolTipText": "14 Shift Duty Span Hour Violation at 5-Oct-16 06:25",
    "accountId": 1248281,
    "id": 2139443,
    "lastChangedDate": "2016-10-05T08:02:14.511Z"
  },
  {
    "userId": 1253061,
    "driverHistoryId": 5155627,
    "timestamp": "2016-11-10T16:07:59.363Z",
    "violationName": "SHIFT_DRIVE_HOURS",
    "iconLabel": "11",
    "message": "You have exceeded your shift driving time of 11 hours",
    "title": "Driving Hours In Shift",
    "toolTipText": "11 Shift Driving Hour Violation at 11-Nov-16 00:07",
```


A fleet of ten trucks may associate six with one type of inspection, and four with another based on the type of truck and relevant inspection points.

DVIRS

Drivers subject to FMCSA regulations are required to perform full vehicle inspections at various times and to record the inspection results in a *Driver-Vehicle Inspection* Report. In the Azuga ELD system, the fields relevant to an inspection are defined by an inspection template. Each *asset* in an account can be associated with a particular template and then grouped in a single DVIR record.

DVIRS are structured as hierarchy. Each DVIR will be associated with one or more asset inspections. Each asset inspection that belong in a DVIR are also associated with one *template*, e.g., “Hauler/Tanker”. A *template* in turn can have one or more *areas* (e.g., Safety Equipment and Vehicle). Finally, each *area* consists of several *items* (e.g., Air Compressor, Hoses, Belts) that constitute the actual parts to be inspected.

The JSON structure returned by the DVIRS API has the following form:

```
{
    "userId": 1253782,
    "startTime": "2017-03-08T08:46:39.643Z",
    "endTime": "2017-03-08T08:48:03.100Z",
    "status": "CERTIFIED",
    "inspectionType": "PRE_TRIP",
    "inspectorType": "DRIVER",
    "location": "",
    "odometerKm": 42.21,
    "certifyMessage": "I, Jack Jill, certify that this DVIR is correct and true.",
    "generalComments": "",
    "equipmentCondition": "SATISFACTORY",
    "signatureMediaId": 36631,
    "dvirFormIds": [
        31108,
        31109
    ],
    "accountId": 1248281,
    "id": 31919,
    "lastChangedDate": "2017-03-08T08:49:58.260Z"
},
```


Values in this structure include

Field	Description
userId	Identifier of the individual performing the DVIR
startTime/endTime	The times at which the inspection was started/completed respectively.
status	Either IN_PROGRESS or CERTIFIED
inspectionType	Reflects whether this inspection was completed PRE_TRIP or POST_TRIP
inspectorType	Either DRIVER or MECHANIC
location	Location where the DVIR was performed
odometerKm	The odometer of the asset when the DVIR was performed.
certifyMessage	Message of the inspector along with the inspector name.
generalComments	General comments from the inspector.
equipmentCondition	Condition of the assets inspected. Can either be DEFECTS_CORRECTED, UNSATISFACTORY or SATISFACTORY.
signatureMediaId	Identifier of the media containing the inspector's signature.
dvirFormIds	Identifiers of the individual inspections per asset that belong to the DVIR record.

Table 5 - Fields returned from DVIRS API

Note that a DVIR record has the dvirFormIds field which are identifiers for each of the inspections that belong to that specific DVIR. In the example above, the DVIR record indicates two asset templates were used in the inspection: 31108 and 31109. Details of the asset template's areas and specific items inspected in the DVIR can be accessed at the dvirForms endpoint:

https://eld.azuga.com/api/v2/dvirForms/31108+31109

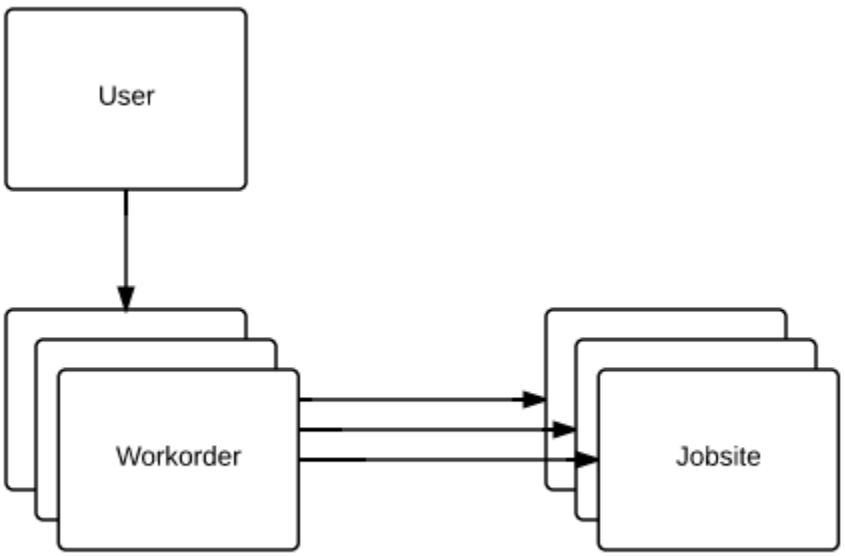
Work Orders

In addition to monitoring Driver Hours of Service for compliance with the FMCSA regulations, Azuga ELD Hours of Service app provides a powerful facility for assigning particular jobs (work orders) to individual drivers. Drivers receive notifications of their assignments on their mobile device and can register the time of completion of the work order.

Each *Work Order* is associated with a particular *Job Site* record as shown in the illustration below.

Each **workorder** is associated with a **user** (referenced by userId) and a **jobsite** (referenced by jobSiteId)

A given **user** may be assigned several **workorders**



Programmatically, the *jobSiteId* field of the structure returned from the Work Orders API request can be used to retrieve information about the corresponding job site.

```
{
    "description": "Work Order 1",
    "userId": 1253937,
    "jobSiteId": 37179,
    "actualStartTime": "2017-01-05T00:15:07.018Z",
    "actualStopTime": "2017-01-05T00:30:59.437Z",
    "completedTime": "2017-01-05T00:31:01.473Z",
    "requestedStartTime": "2017-01-05T00:12:00.000Z",
    "viewTimestamp": "2017-01-05T00:14:58.309Z",
    "signature": "BLOB of signature",
    "totalTimeWorked": "PT952.427S",
    "fields": {
        "__WORK_ORDER_ON_DEVICE": "true"
    },
    "media": [
        {
            "name": "IMG_2017-01-05-08.30.19.jpg",
            "filename": "IMG_2017-01-05-08.30.19.jpg",
            "id": 28197
        }
    ],
    "accountId": 1248281,
    "id": 21729,
    "lastChangedDate": "2017-01-05T00:30:29.869Z"
},
```


Job Sites

As noted above, *Job Sites* are used in conjunction with *Work Orders* and provide simple location information about a particular business or residence.

For example, to retrieve information about a specific job site (in this case the site with the id 37179) you can issue this API request:

https://eld.azuga.com/api/v2/jobSites/37179


```
[
    {
        "fullAddress": "505 N Michigan Ave, Chicago, IL 60611, USA",
        "name": "Default",
        "street": "505 North Michigan Avenue",
        "city": "Chicago",
        "state": "Illinois",
        "postalCode": "60611",
        "country": "United States",
        "location": "POINT (-87.6235815 41.8911622)",
        "note": "123",
        "fields": {},
        "accountId": 1248281,
        "id": 37179,
        "lastChangedDate": "2016-08-08T09:25:48.176Z"
    },
```


The field names should be largely self-explanatory with the possible exception of the "note" field which presents the free form text entered by the user when the job site was defined.

Getting Started with Sample Applications

Azuga ELD provides sample code to illustrate how the RESTful calls can be used to retrieve data from and send data to the backend system. Almost all samples are written in JavaScript and can be downloaded from the " developer.eldazuga.com" website. They can be launched either from a web server environment or directly from a file system. The Sample Applications link can be found here:

http://developer.eldazuga.com/index.php/sample-applications-v2/

Basic "read" Access to Objects

Probably the simplest way to begin working with the API set is to explore the various endpoints using the http "GET" operator. The file "Azuga ELD_API_Testbed-V2.html" is a small JavaScript application that, when launched, provides an interface to retrieve all records of a certain type within an account. The application, shown in the screenshot below, provides buttons that correspond to several of the objects exposed in the API.

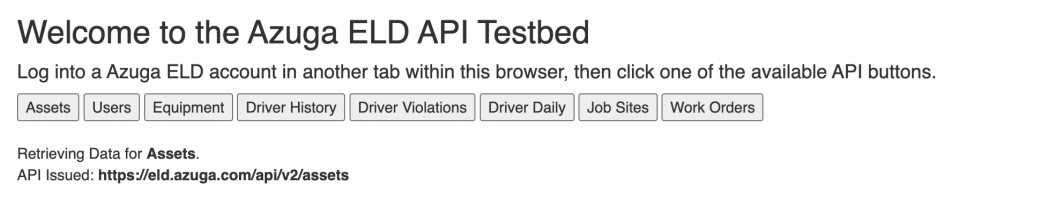


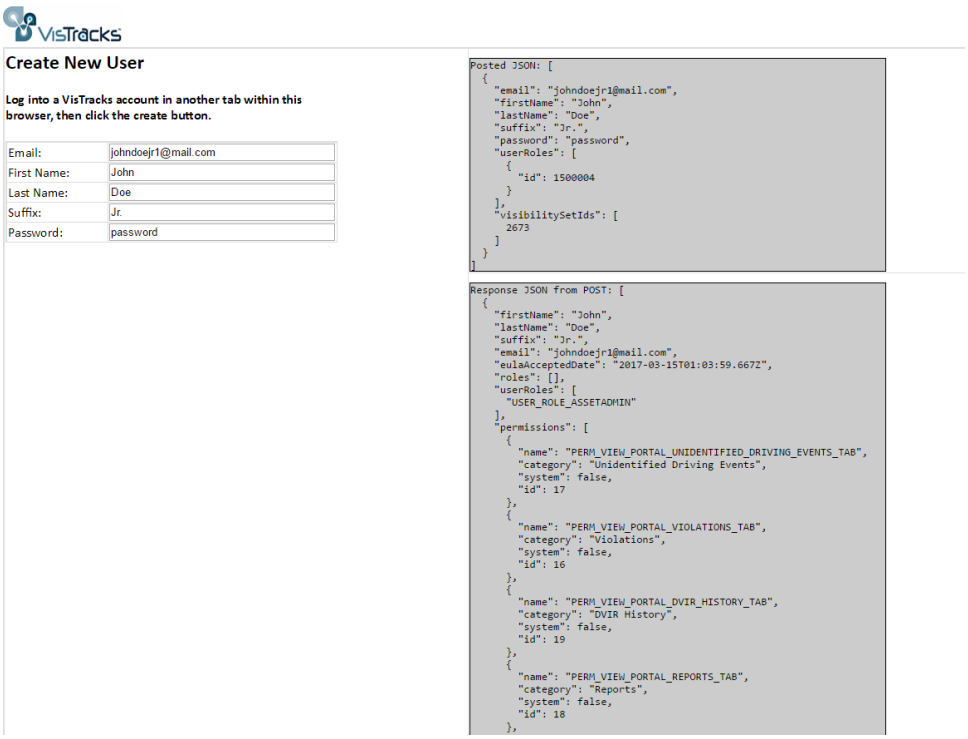
Image 2 - interface from Azuga ELD_API_Testbed-V2 sample

Note that the actual URL used to retrieve the information is presented so it can be easily copied and used in any additional experimentation.

Creating Objects

The next set of samples illustrates how to create new instances of various objects. These samples, along with an explanatory "ReadMe", are included in the "createSamples.zip" archive on the developer.eldazuga.com site.

This zip file expands into a directory containing several HTML pages with names of the form "create*Type*.html. Corresponding to each of these pages is a small JavaScript file with the same basename, but with a ".js" suffix. The real logic that is common to all the pages resides in the createxxx-jscore.js file. As long as these files reside in the same directory you should be able to launch any sample application by clicking on the ".html" file to open it in a browser. For example, the illustrations below show the "createAsset" application both before and after issuing the POST to create the object. The body of the POST sent to the server, as well as that of the reply from the server are displayed after the create operation.



Summary

This document discussed several of the key objects that form the basis of the Azuga ELD platform. More detailed information about the individual APIs is presented on the "developer.eldazuga.com" website.The table below provides a quick review of the objects covered in this present discussion. Depending on the particular account, <baseURL> will be one of the following:

https://eld.azuga.com

or

https://staging.eld.azuga.com

Object	Description and REST access point
Users	Provides access to one or more users associated with the target account. **<baseURL>/api/v2/users**
Assets	End-user facing abstraction for all data collected about a particular vehicle **<baseURL>/api/v2/assets**
Drivers	Users that login to the application and perform driving events **<baseURL>/api/v2/drivers**
Driver Daily	Record that summarizes the activities of a driver using the HOS application. **<baseURL>/api/v2/driverDailies**
Driver History	Record that indicates time and place of explicit driving status changes **<baseURL>/api/v2/driverHistories**
Driver Violations	Records that reflect any driving thresholds that were exceeded **<baseURL>/api/v2/driverViolations**
Driver Status	Records summarizing current driving state for each driver. **<baseURL>/api/v2/driverStatuses**
DVIRS	Records that capture the state of all asset inspections and identifiers to the individual asset templates with areas and points. **<baseURL>/api/v2/dvirs**
Jobsites	Address information for locations where particular activities are to be performed by a mobile worker **<baseURL>/api/v2/jobSites**
Work Orders	Record that defines a particular assignment for a user within the system **<baseURL>/api/v2/workOrders**

Table 6 - Summary of important APIs