Programmatic Access to the VisTracks Platform

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

Version 4

VisTracks, Inc.
June 26, 2015

Contents

Introduction 2

Prerequisites 3

Accessing the VisTracks REST APIs 3

- Example API Request 4

Fundamental Objects 6

- Users 6
- Devices 8
- Assets 9
- Provision Records 9
- Configuring Smartphone and Tablet Objects 9

Drivers and Hours of Service Objects 10

- DriverDaily 11
- DriverHistory 13
- DriverViolations 15
- Inspections 16
- Workorders 17
- Jobsites 18

Getting Started with Sample Applications 19

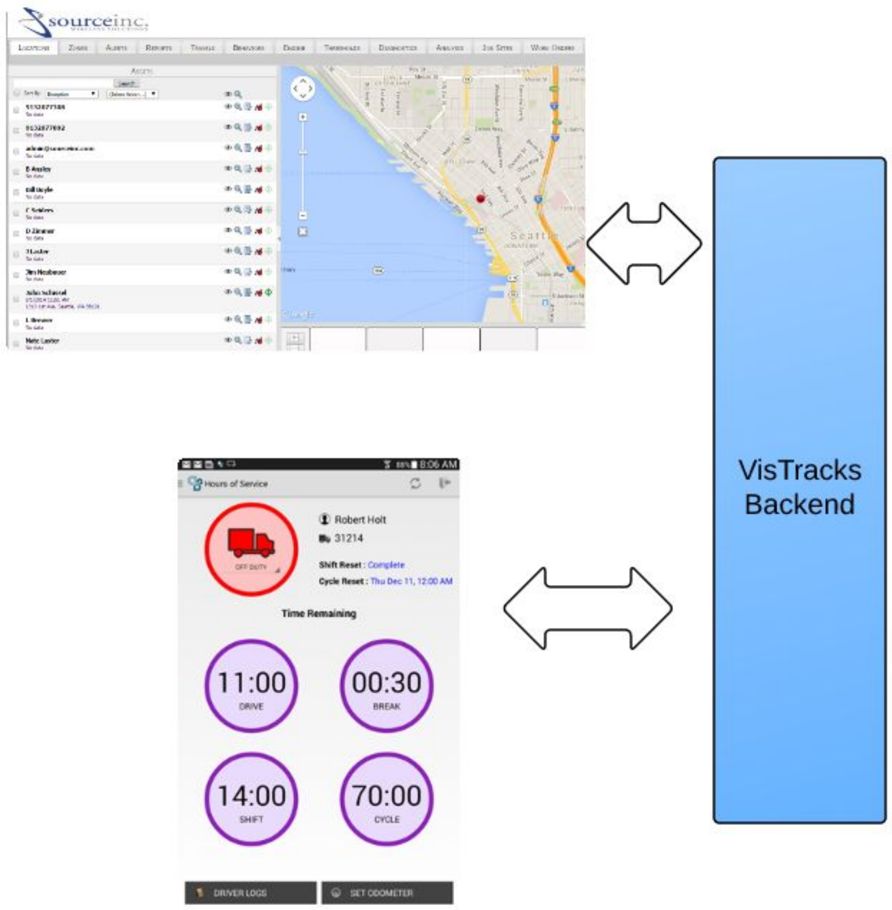
- Basic “read” Access to Objects 19
- Creating Objects 19
- Updating, Deleting and Combining Objects 20

Summary 21

Introduction

VisTracks 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 VisTracks, 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.vistracks.com" site.



VisTracks 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 VisTracks APIs, it's necessary to have obtained an account on one of the VisTracks servers -- typically either

http://live.vistracks.com

http://demo.vistracks.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 VisTracks REST APIs

Authenticating REST requests

VisTracks 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 an XML or 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,

http://live.vistracks.com/api/v1/assets

would return all assets in the account, while

http://live.vistracks.com/api/v1/assets/143525

would return a single element from the collection.

Because VisTracks 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, almost all the APIs discussed in this document manipulate data in an XML format. However, plans going forward include a transition to a JSON format instead. There is currently one such example -- the "DriverStatus" API discussed later in this document will return information in 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 VisTracks 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 "vistracks.com" site in a separate browser tab and issue any API calls from another tab in the same browser. In this arrangement, the authenticated "sessionid" is automatically sent to the VisTracks 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 VisTracks portal (e.g., live.vistracks.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)

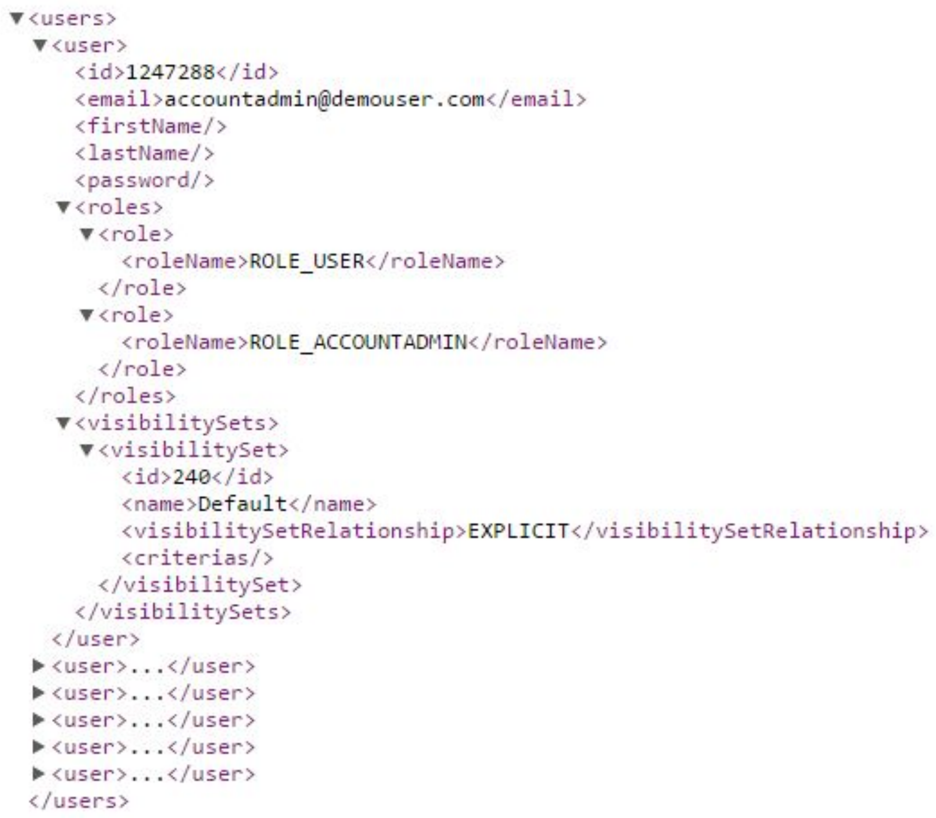
2. Open a new tab in the same browser

3. From this new tab, enter the URL for an API, e.g.,

http://live.vistracks.com/api/v1/user

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

XML returned by a "GET" request


```
▼<users>
  ▼<user>
     <id>1247288</id>
     <email>accountadmin@demouser.com</email>
     <firstName/>
     <lastName/>
     <password/>
   ▼<roles>
     ▼<role>
        <roleName>ROLE_USER</roleName>
      </role>
     ▼<role>
        <roleName>ROLE_ACCOUNTADMIN</roleName>
      </role>
    </roles>
   ▼<visibilitySets>
     ▼<visibilitySet>
        <id>240</id>
        <name>Default</name>
        <visibilitySetRelationship>EXPLICIT</visibilitySetRelationship>
        <criterias/>
      </visibilitySet>
    </visibilitySets>
  </user>
 ▶<user>...</user>
 ▶<user>...</user>
 ▶<user>...</user>
 ▶<user>...</user>
 ▶<user>...</user>
</users>
```

You can try accessing other objects, e.g., *assets*, in a similar fashion. (http://live.vistracks.com/api/v1/assets)

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

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

To perform actions other than "GET", (e.g., creating or modifying 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:

http://live.vistracks.com/api/v1/user	returns information about all users
http://live.vistracks.com/api/v1/user/1246844	returns information about a single user with the supplied id.
http://live.vistracks.com/api/v1/user/1246844+1246849	returns information about two users
http://live.vistracks.com/api/v1/user?role=ROLE_USER	returns information about all users with the given “role”.

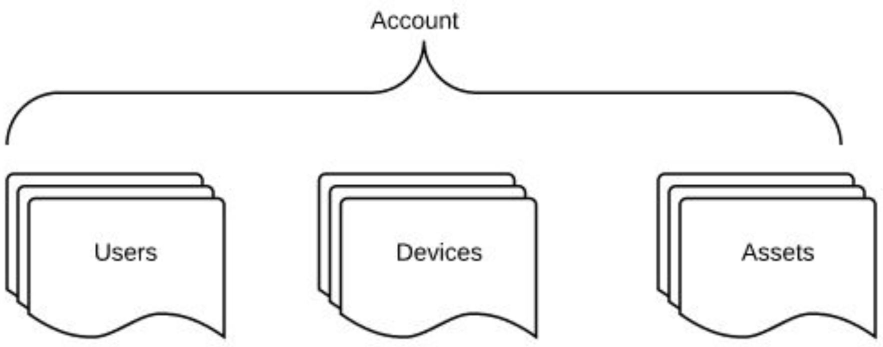
Fundamental Objects

In order to effectively use and manipulate the objects exposed by the VisTracks 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
- Assets
- Devices

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 address.

One or more users can be associated with a given account, each of whom is uniquely 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>” tag and has the value 1247288.) 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>1248584</userId>
```

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 *roles.* A user's roles determine what capabilities they have within the system as summarized in the table below. In the above XML example, the user was designated as having ROLE_ACCOUNTADMIN, and ROLE_USER.

	Create and manage top-level accounts	Create and manage sub-accounts	Create and manage assets and devices	Create and manage users	Manage site branding and account properties	Run available applications	Modify the layout of "Silverlight" applications	Schedule "offline" reports	Able to be scheduled for work orders
ROLE_ACCOUNTADMIN		🟩	🟩	🟩	🟩	🟩	🟩	🟩	
ROLE_ACCOUNTCREATOR	🟩		🟩			🟩		🟩	
ROLE_ASSETADMIN			🟩			🟩		🟩	
ROLE_USERADMIN				🟩		🟩		🟩	
ROLE_DASHBOARDAUTHOR						🟩	🟩	🟩	
ROLE_ELEVATEDUSER						🟩		🟩	
ROLE_USER						🟩			
ROLE_WORKORDERASSIGNEE						🟩			🟩

An "Account Admin" for a top-level account can create, modify, and delete sub-accounts. An "Account Admin" for a sub-account has full administrative access for that sub-account, but no ability to create any additional sub-accounts.

"User" and "ElevatedUser" are equivalent at present, but reserved for future use by applications.

Devices

Devices represent a physical instrument capable of reporting position and other information to the VisTracks system. For this discussion, devices can be broadly divided into two categories: those that correspond to a smartphone or tablet, and those that are distinct physical telematics devices that may, for example, plug into the vehicle bus in a car or truck. Any examples that relate to mobile workers or driver HOS data will be using devices from the former group -- tablets or smartphones.

Information about devices in an account can be retrieved through the

http://live.vistracks.com/api/v1/devices api.

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 *device*). The collected data, though, is conceptually associated with the asset and from an API perspective will be retrieved by referencing the *asset* object.

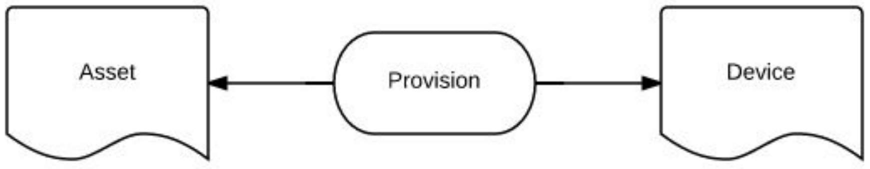
In the standard VisTracks clients (SnapTraq web portal or Android/iOS app) the "Asset Name" is typically used to identify a given piece of equipment. Neither the name given to the supporting device, nor its unique identifier are usually visible to the end user.

Provision Records

As noted in the previous couple sections, the VisTracks system distinguishes between objects being monitored (assets) and the telematics equipment doing the monitoring (devices.) A third object known as a *provision record* provides the association between these two concepts. Provision records offer a time-bound mapping between device and asset. This allows the system to report a consistent history of a over a period.

Provisions provide the association between *devices* and *assets.*

To further illustrate this concept, imagine a situation in which a Vehicle A was brought on line at the beginning of February using a particular OBD device. A month later, that OBD device failed and was replaced by a different unit. There would be two provision records for Vehicle A -- one covering the month of February, the second covering the time from March forward.



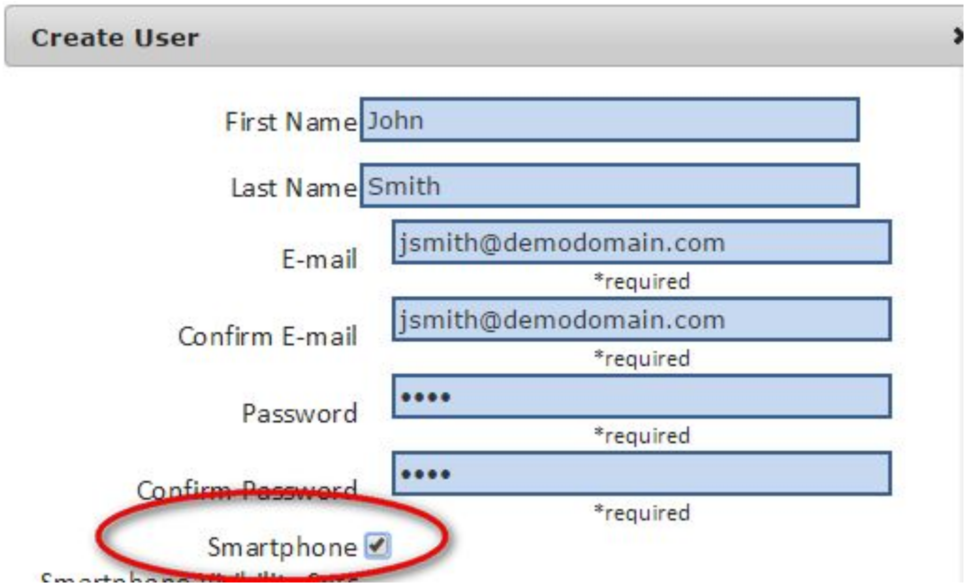
Provision information is available through the "provision" API:

http://live.vistracks.com/api/v1/provision

Configuring Smartphone and Tablet Objects

Devices of type "phone" (which include both smartphone and tablet units) warrant some additional explanation in that, unlike most telematics devices, they have an implicit relation to "users" within the VisTracks system. This is because while most devices use some attribute inherent to the hardware (commonly the IMEI or MEID) as their unique identifier in the system, the class of "phone" devices rely on the email address assigned as part of a user's credentials as that device's unique identifier. To facilitate this, the administrative pages that support the creation of a new user include an option to mark an entry as a "smartphone user". (See illustration below.)

Setting up a new "smartphone" user will implicitly create a corresponding device and asset based on teh user's assigned email address.


Create User

First Name	John
Last Name	Smith
E-mail	jsmith@demodomain.com *required
Confirm E-mail	jsmith@demodomain.com *required
Password	•••• *required
Confirm Password	•••• *required
Smartphone	✓

By checking this option when adding a new user, the system will automatically create not only the requested user record, but in addition,

a) an Asset record whose "name" is given by the first and last names provided in the user record,

b) a Device of type "phone" whose unique identifier is the supplied email address, and

c) a Provision record to associate the Asset to the Device.

Drivers and Hours of Service Objects

The preceding material discussed some of the basic data constructs underlying the VisTracks system. The concepts of users, devices, 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 "SnapTraq" app offered by VisTracks 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 VisTracks 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 commerical vehicle subject to the FMSCA regulations.
Driver Status	A read-only 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
Inspections	Details of the items inspected on a particular vehicle

Table 1 - APIs related to Hours of Service application

Pictorially, the relationships among various items is shown below:

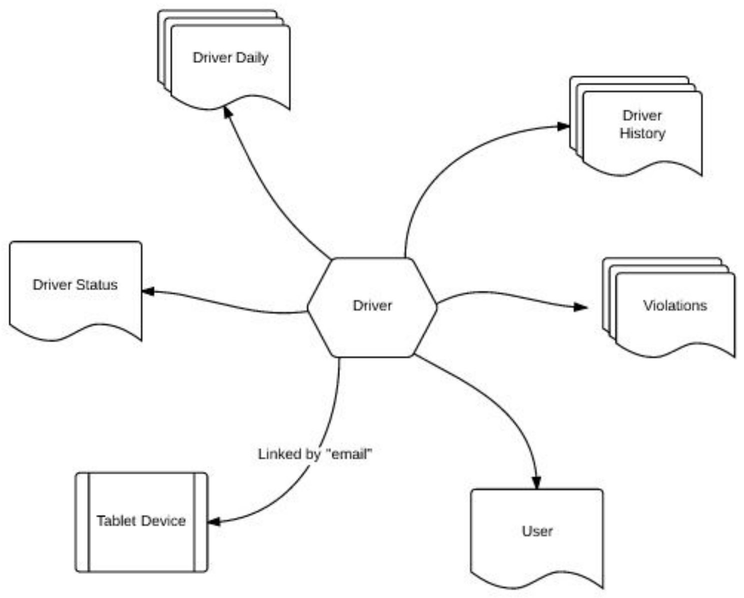


Image 1 - Relationships between a Driver object and associated data

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

DriverDaily

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:

http://live.vistracks.com/api/v1/driverdaily?allUsers=true&fromDate=2015-03-10&toDate=2015-03-10&includeLatestStatus=true

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

-allUsers=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:

http://live.vistracks.com/api/v1/driverdaily/?userId=1248594&fromDate=2015-03-10&toDate=2015-03-10&includeLatestStatus=true

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.

-fromDate, toDate define the time range of interest.

-includeLatestStatus is a boolean field that, if true, will include information corresponding to the latest report from the device (driver location and status) that falls within the specified time range.

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

```
<driverDailies>
    <driverDaily>
      <id>745</id>
      <userId>1248518</userId>
      <carrier>Joes Trucking</carrier>
      <certified>false</certified>
      <coDriverName>John Doe</coDriverName>
      <cycle>US60hr7days</cycle>
      <date>2015-01-27</date>
      <driverEmail>admin@hcss.com</driverEmail>
      <driverName>[Driver Name]</driverName>
      <driverPhone/>
      <exceptions>None</exceptions>
      <timeZone>EST</timeZone>
      <homeTerminalAddress>100 Main Street</homeTerminalAddress>
      <mainOfficeAddress>801 Warrenville Road, Suite 50, Lisle, IL
      60532</mainOfficeAddress>
      <beginOdometer>0</beginOdometer>
      <endOdometer>0</endOdometer>
      <odometerUnits>MILES</odometerUnits>
      <shippingDocsManifestNo>Shipping Document
      100</shippingDocsManifestNo>
      <shippingDocsShipperCommodity>[Shipper or
      Commodity]</shippingDocsShipperCommodity>
      <trailerId>[Trailer Id]</trailerId>
      <truckId>[Truck Id]</truckId>
      <fields/>
      <lastChangedDate>2015-01-27T18:26:18.615Z</lastChangedDate>
    </driverDaily>
```

```
</driverDailies>
```

Principal values 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
date	Date of Report
driverEmail	Based on *user* object associated with person logged into tablet
beginOdometer	Value entered by driver at start of day
endOdometer	Value entered by driver at endof day
odometerUnits	MILES or KM
truckId	Identifier of truck entered in driver settings
trailerId	Identifier of trailer entered in driver settings
Filename	name of certified log in PDF format

Table 2 – Sample fields returned from DriverDaily Request

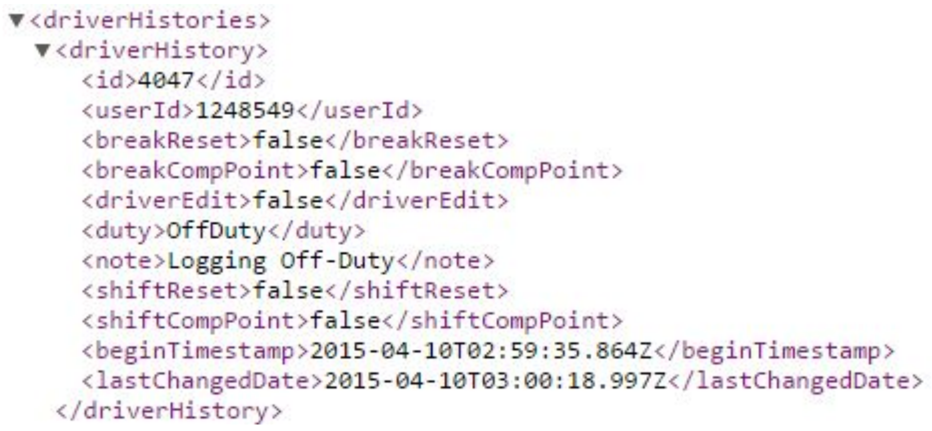
DriverHistory

Drivers using the SnapTraq tablet 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.

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 XML appears as follows:



```
▼<driverHistories>
  ▼<driverHistory>
     <id>4047</id>
     <userId>1248549</userId>
     <breakReset>false</breakReset>
     <breakCompPoint>false</breakCompPoint>
     <driverEdit>false</driverEdit>
     <duty>OffDuty</duty>
     <note>Logging Off-Duty</note>
     <shiftReset>false</shiftReset>
     <shiftCompPoint>false</shiftCompPoint>
     <beginTimestamp>2015-04-10T02:59:35.864Z</beginTimestamp>
     <lastChangedDate>2015-04-10T03:00:18.997Z</lastChangedDate>
   </driverHistory>
```

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
duty	new driving state – one of OffDuty, OnDuty, SleeperBerth, Driving
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.
beginTimestmap	date and time stamp recording time when state change was initiated
breakCompPoint	Boolean indicating if the “break” timer was reset to max value.
shiftCompPoint	Boolean indicating if the “shift” timer was reset to max value.
cycleCompPoint	Boolean indicating if the “cycle” timer was reset to max value.
lastChangedDate	time at which the most recent state change for the driver was recorded.

Table 3 - fields returned by the DriverHistory API

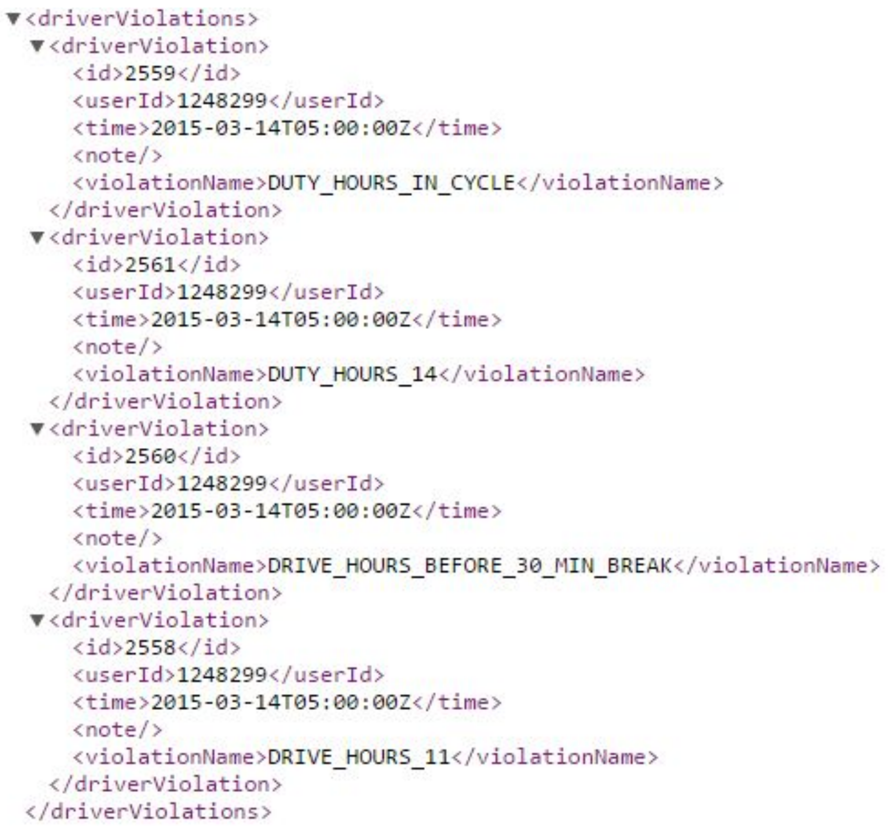
DriverViolations

The “DriverViolations” API includes a list of any violations incurred over a specified time range. The returned XML structure takes the form shown in the excerpt below. Each violation includes a timestamp reflecting the date and time at which the violation occurred and in indication of the type of violation. Possible violations include

Violation Type	Description
DUTY_HOURS_IN_CYCE	Driver exceeded allowed “on-duty” hours in the current cycle
DUTY_HOURS_14	Driver exceeded allowed driving time in a 14 hour period
DRIVE_HOURS_11	Driver exceeded 11 hour limit without a break
DRIVE_HOURS_BEFORE_30_MIN_BREAK	Driver did not take required break after continuous hours of driving
DRIVE_HOURS__TO_BREAK	Driver did not take required break after 8 continuous hours of driving
SHIFT_DUTY_HOURS	Driver exceeded maximum duty hours allowed on shift
SHIFT_DRIVE_HOURS	Driver exceeded maximum driving hours allowed on shift

Table 4 - Possible Values of Driver Violations

Violations are returned in an XML collection as illustrated below. Not that each violation has its own unique id, but all refers to the userId of the associated driver.



```
▼<driverViolations>
  ▼<driverViolation>
     <id>2559</id>
     <userId>1248299</userId>
     <time>2015-03-14T05:00:00Z</time>
     <note/>
     <violationName>DUTY_HOURS_IN_CYCLE</violationName>
   </driverViolation>
  ▼<driverViolation>
     <id>2561</id>
     <userId>1248299</userId>
     <time>2015-03-14T05:00:00Z</time>
     <note/>
     <violationName>DUTY_HOURS_14</violationName>
   </driverViolation>
  ▼<driverViolation>
     <id>2560</id>
     <userId>1248299</userId>
     <time>2015-03-14T05:00:00Z</time>
     <note/>
     <violationName>DRIVE_HOURS_BEFORE_30_MIN_BREAK</violationName>
   </driverViolation>
  ▼<driverViolation>
     <id>2558</id>
     <userId>1248299</userId>
     <time>2015-03-14T05:00:00Z</time>
     <note/>
     <violationName>DRIVE_HOURS_11</violationName>
   </driverViolation>
 </driverViolations>
```

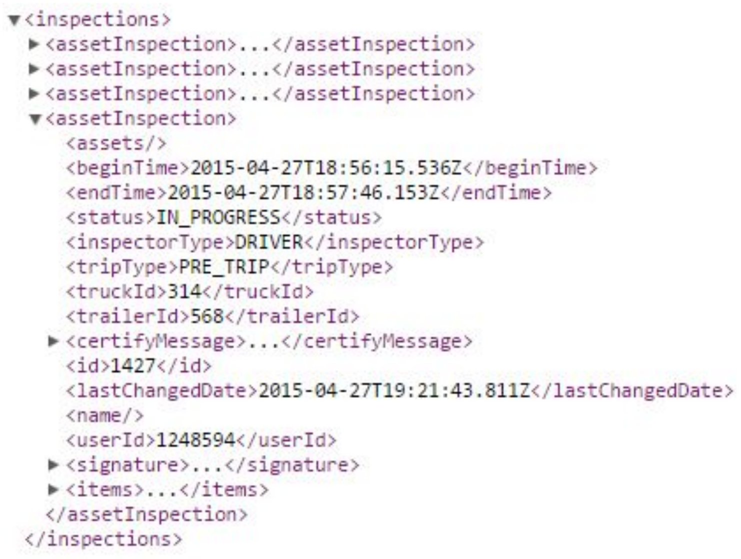
Inspections

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.

Drivers subject to HOS regulations are required to perform full vehicle inspections at various times. In the VisTracks 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.

Inspection templates are structured as hierarchy. Each inspection will be associated with one *category*, e.g., “Hauler/Tanker”. A *category* 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 XML structure returned by the “inspections” API has the following form:



```
▼<inspections>
  ▶<assetInspection>...</assetInspection>
  ▶<assetInspection>...</assetInspection>
  ▶<assetInspection>...</assetInspection>
  ▼<assetInspection>
     <assets/>
     <beginTime>2015-04-27T18:56:15.536Z</beginTime>
     <endTime>2015-04-27T18:57:46.153Z</endTime>
     <status>IN_PROGRESS</status>
     <inspectorType>DRIVER</inspectorType>
     <tripType>PRE_TRIP</tripType>
     <truckId>314</truckId>
     <trailerId>568</trailerId>
    ▶<certifyMessage>...</certifyMessage>
     <id>1427</id>
     <lastChangedDate>2015-04-27T19:21:43.811Z</lastChangedDate>
     <name/>
     <userId>1248594</userId>
    ▶<signature>...</signature>
    ▶<items>...</items>
   </assetInspection>
 </inspections>
```

Values in this structure include

Field	Description
beginTime/endTime	The times at which the inspection was started/completed respectively.
status	Either IN_PROGRESS or CERTIFIED
inspectorTrype	Either DRIVER or MECHANIC
tripType	Reflects whether this inspection was completed PRE_TRIP or POST_TRIP
truckId/trailerId	Asset names of the truck/trailer combo being inspected
certifyMessage	A string presented to the driver to explicitly acknowledge that they have completed the inspection.
signature	An ASCII encoding of the driver's signature image captured on the tablet.
userId	reference to the individual performing the inspection
items	A collection of the individual inspection points examined as part of the inspection.

Table 5 - Fields returned from inspections API

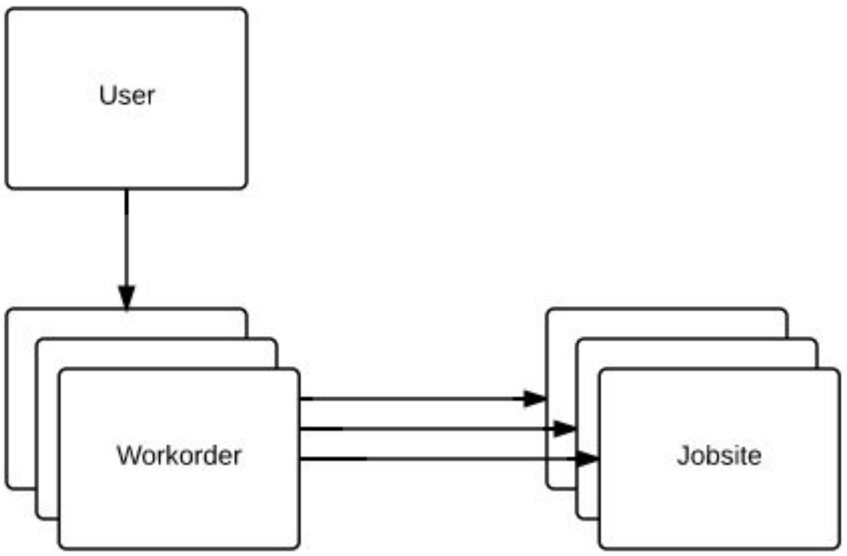
Workorders

In addition to monitoring Driver Hours of Service for compliance with the FMCSA regulations, SnapTraq provides a powerful facility for assigning particular jobs (workorders) to individual drivers. Drivers receive notification of their assignments on their mobile device and can register the time of completion of the workorder.

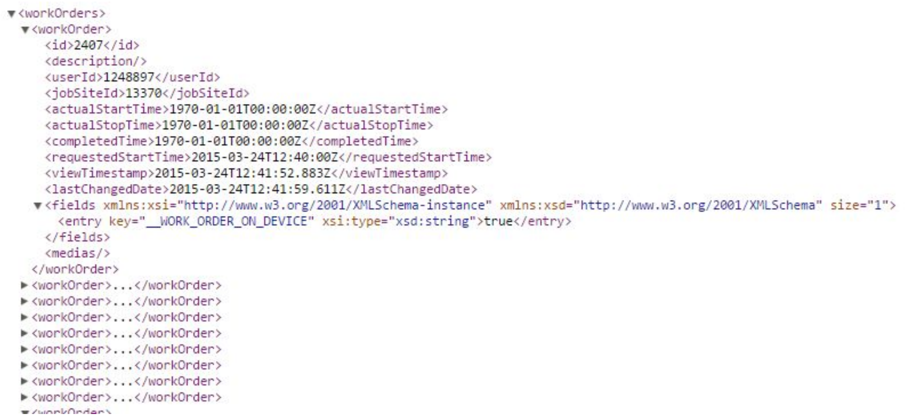
Each *workorder* is associated with a particular *jobsite* record as shown in the illustration below.

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 “workorders” API request can be used to retrieve information about the corresponding job site.



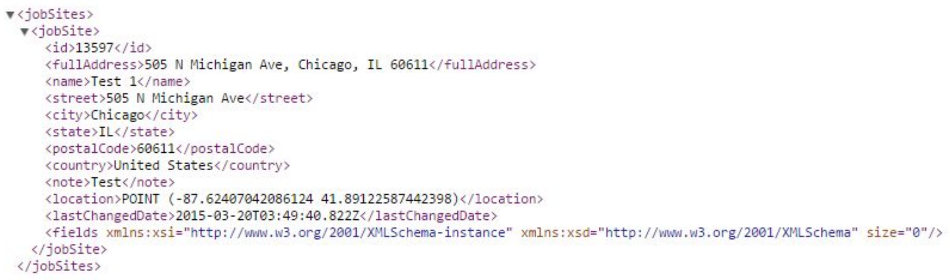
```
▼<workOrders>
  ▼<workOrder>
     <id>2407</id>
     <description/>
     <userId>1248897</userId>
     <jobSiteId>13370</jobSiteId>
     <actualStartTime>1970-01-01T00:00:00Z</actualStartTime>
     <actualStopTime>1970-01-01T00:00:00Z</actualStopTime>
     <completedTime>1970-01-01T00:00:00Z</completedTime>
     <requestedStartTime>2015-03-24T12:40:00Z</requestedStartTime>
     <viewTimestamp>2015-03-24T12:41:52.883Z</viewTimestamp>
     <lastChangedDate>2015-03-24T12:41:59.611Z</lastChangedDate>
    ▼<fields xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" size="1">
       <entry key="__WORK_ORDER_ON_DEVICE" xsi:type="xsd:string">true</entry>
     </fields>
     <medias/>
   </workOrder>
  ▶<workOrder>...</workOrder>
  ▶<workOrder>...</workOrder>
  ▶<workOrder>...</workOrder>
  ▶<workOrder>...</workOrder>
  ▶<workOrder>...</workOrder>
  ▶<workOrder>...</workOrder>
  ▶<workOrder>...</workOrder>
  ▶<workOrder>...</workOrder>
  ▼<workOrder>
```

Jobsites

As noted above, *jobsites* are used in conjunction with *workorders* 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 13597) you can issue this API request:

http://live.vistracks.com/api/v1/jobsites/13597



```
▼<jobSites>
  ▼<jobSite>
     <id>13597</id>
     <fullAddress>505 N Michigan Ave, Chicago, IL 60611</fullAddress>
     <name>Test 1</name>
     <street>505 N Michigan Ave</street>
     <city>Chicago</city>
     <state>IL</state>
     <postalCode>60611</postalCode>
     <country>United States</country>
     <note>Test</note>
     <location>POINT (-87.62407042086124 41.89122587442398)</location>
     <lastChangedDate>2015-03-20T03:49:40.822Z</lastChangedDate>
     <fields xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" size="0"/>
   </jobSite>
 </jobSites>
```

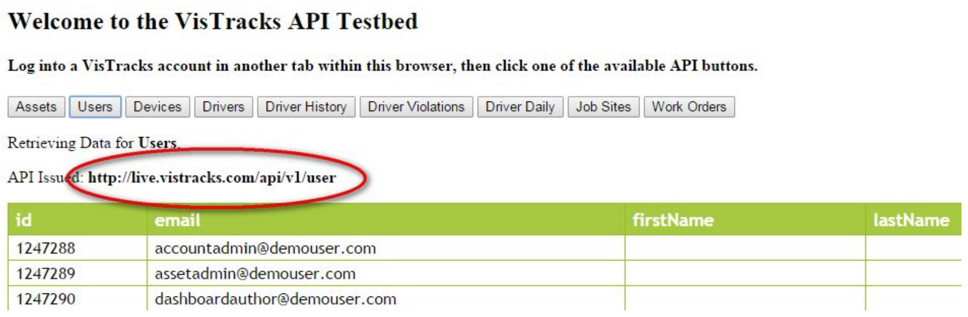
The field names should be largely self-explanatory with the possible exception of the "note" field which presents the free form text entered by the user when the job site was defined.

Getting Started with Sample Applications

VisTracks 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.vistracks.com" website. They can be launched either from a web server environment or directly from a file system.

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 "vt_api_testbed.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.



Welcome to the VisTracks API Testbed

Log into a VisTracks account in another tab within this browser, then click one of the available API buttons.

Retrieving Data for **Users**

API Issued: **http://live.vistracks.com/api/v1/user**

id	email	firstName	lastName
1247288	accountadmin@demouser.com		
1247289	assetadmin@demouser.com		
1247290	dashboardauthor@demouser.com		

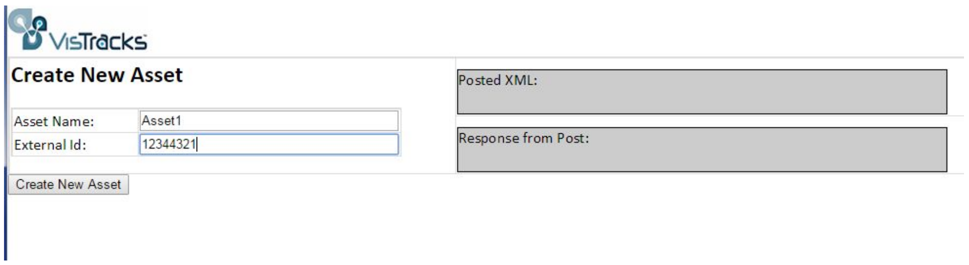
Image 2 - interface from vt_api_testbed sample

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.vistracks.com site.

This zipfile 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 shows 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.

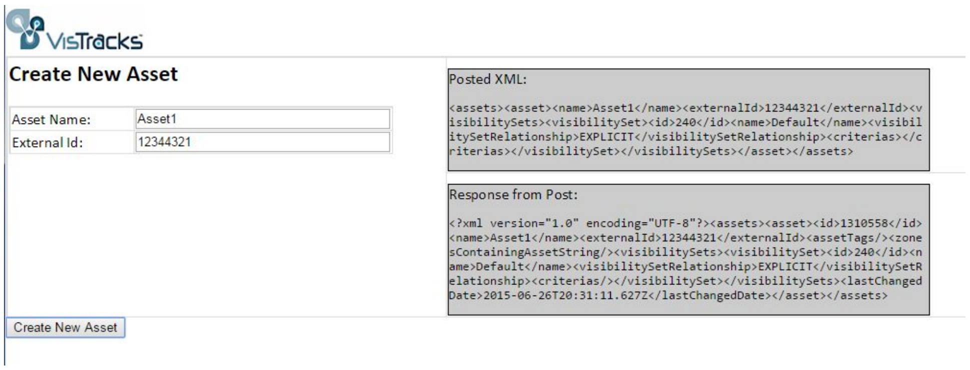


Create New Asset

Asset Name:	Asset1
External Id:	12344321

Posted XML:

Response from Post:



Create New Asset

Asset Name:	Asset1
External Id:	12344321

Posted XML:

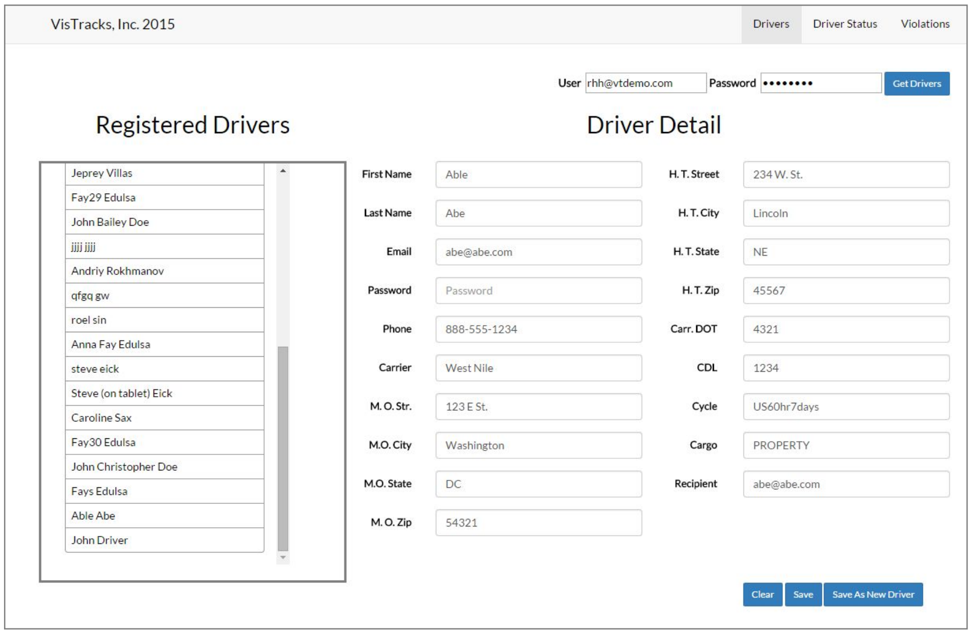
```
<assets><asset><name>Asset1</name><externalId>12344321</externalId><v
isibilitySets><visibilitySet><id>240</id><name>Default</name><visibil
itySetRelationship>EXPLICIT</visibilitySetRelationship><criterias></c
riterias></visibilitySet></visibilitySets></asset></assets>
```

Response from Post:

```
<?xml version="1.0" encoding="UTF-8"?><assets><asset><id>1310558</id>
<name>Asset1</name><externalId>12344321</externalId><assetTags/><zone
sContainingAssetString/><visibilitySets><visibilitySet><id>240</id><n
ame>Default</name><visibilitySetRelationship>EXPLICIT</visibilitySetR
elationship><criterias/></visibilitySet></visibilitySets><lastChanged
Date>2015-06-26T20:31:11.627Z</lastChangedDate></asset></assets>
```

Updating, Deleting and Combining Objects

The final set of sample applications provide a somewhat more complete "reference" framework for working with the driver-related objects in particular.   As shown in the capture below, for example, the application will provide a means to create and manipulate drivers and their associated records such as Violations, Status, History, etc.   These samples are located in the "sampleDevSite" folder.



User rhh@vtdemo.com Password ••••••••

Registered Drivers

Jeprey Villas
Fay29 Edulsa
John Bailey Doe
jjjj jjjj
Andriy Rokhmanov
qfgq gw
roel sin
Anna Fay Edulsa
steve eick
Steve (on tablet) Eick
Caroline Sax
Fay30 Edulsa
John Christopher Doe
Fays Edulsa
Able Abe
John Driver

Driver Detail

First Name	Able	H. T. Street	234 W. St.
Last Name	Abe	H. T. City	Lincoln
Email	abe@abe.com	H. T. State	NE
Password	Password	H. T. Zip	45567
Phone	888-555-1234	Carr. DOT	4321
Carrier	West Nile	CDL	1234
M. O. Str.	123 E St.	Cycle	US60hr7days
M.O. City	Washington	Cargo	PROPERTY
M.O. State	DC	Recipient	abe@abe.com
M. O. Zip	54321		

Summary

This document discussed several of the key objects that form the basis of the VisTracks platform. More detailed information about the individual APIs is presented on the "developer.vistracks.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

http://live.vistracks.com, or

http://demo.vistracks.com

Object	Description and REST access point
User	Provides access to one or more users associated with the target account. **<baseURL>/user**
Asset	End-user facing abstraction for all data collected about a particular vehicle **<baseURL>/asset**
Device	Tablet, Smartphone, or telematics device that captures realtime information about an asset and transmits it to the VisTracks backend **<baseURL>/device**
Provision	Records that define an association between Asset and Device **<baseURL>/provision**

DriverDaily	Record that summarizes the activities of a driver using the HOS application. **<baseURL>/driverdaily**
DriverHistory	Record that indicates time and place of explicit driving status changes **<baseURL>/driverhistory**
DriverViolations	Records that reflect any driving thresholds that were exceeded **<baseURL>/driverviolation**
DriverStatus	Records summarizing current driving state for each driver. **<baseURL>/driverstatus**
Inspections	Records that capture the state of all vehicle inspection points. **<baseURL>/inspections**
Jobsite	Address information for locations where particular activities are to be performed by a mobile worker **<baseURL>/jobsites**
Workorder	Record that defines a particular assigment for a user within the system **<baseURL>/workorders**

Table 6 - Summary of available APIs