AssetLink DeviceManager Web API

Revision 2, May 2018

AssetLink Global LLC

www.assetlinkglobal.com

service@assetlinkglobal.com

+1 (303) 862.8745

Fax: +1 (720) 862.3515

Revision History

Revision	Date of Release	Purpose
Initial Draft	9/16/2016	Based on Portal Fundamental API, as implemented
1	3/3/2017	Updated following lessons-learned from original implementation. Changed name from Data Portal to DeviceManager.
2	5/16/2018	Removed deprecated Device API commands.

Introduction

The AssetLink DeviceManager is the cloud-based server that forms the logical interface between AssetLink customers, and their remotely monitored assets. The over-the-air (OTA) formats for interacting with devices like the AssetPack 3, are necessarily compact and rely on a great deal of context and shared format information. Over the Web, conversely, data and their context can both be shared — for example, a geographic position might be sent over the air as the compressed but obtuse 48-bit value 0x375b51c9375a, but a Web API has the luxury of offering this as {"Lat":38.92268,"Lon":-77.03969}, increasing comprehensibility and reducing the possibility of gross errors in parsing.

This API is how customers can have their own server systems query the DeviceManager, receiving information and sending commands.

It also functions as a mechanism for *data normalization*, where different remote devices, from different sources in different applications with different over-the-air protocols, can have their common information shared in a common way. One unit in the field may be a simple tracking device using Iridium, another may be a sophisticated medicine-temperature-sensing device using cellular, but they are both reporting position, and that position information comes through in a common way from both of them via this API.

This API recognizes that in general, requests into the Manager wish to *select* some set of objects, and then *do* something with them. *Select* a bunch of remote devices, and *send a schedule* to all of them. *Select* a series of Moments, and *get the Points* for all of them. *Select* all the children of a Grp, and *put an Alert on any of them that have gone quiet*. Some of these seem to lend themselves to REST terminology, but it ends up being a little tangential to the idea of GET / PUT / POST / DELETE; with actual pieces of hardware involved, it's more clearly *select* and then *do*.

Groups, Devices, Moments, and Points

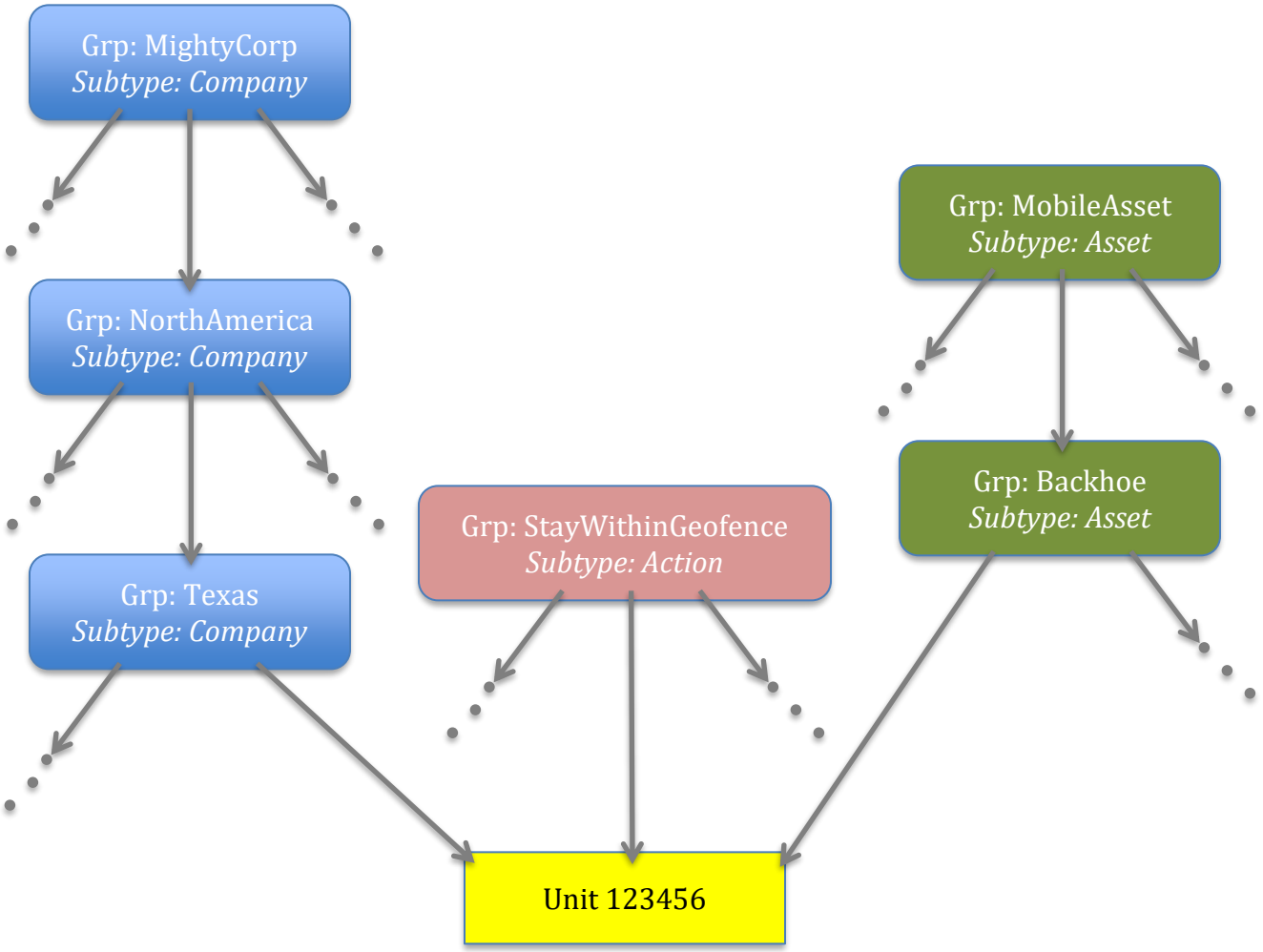
The AssetLink DeviceManager takes in the data provided by any backhaul provider, in whatever format, using whatever protocol, responding with whatever handshake, that each provider defines. It takes these incoming data and reduces them to a common structure:

- a set of key:value pairs (such as Lat:34.56, Lon:-77.02, Door:open),
- which are gathered into associated structures we call *Points* (a PointLocation with Lat and Lon, a PointAlarm saying Hey the door is open),
- which are then collected into a structure called a *Moment*, whose job is to say "At this time, we learned this information",
- that Moment is added to the *Device* that just reported this information,
- the Device belonging to one or more Groups. Groups can describe what part of the corporate organization the unit belongs to; or what type of asset the unit is attached to; or a behavior that the unit should display. Because "Group" is a

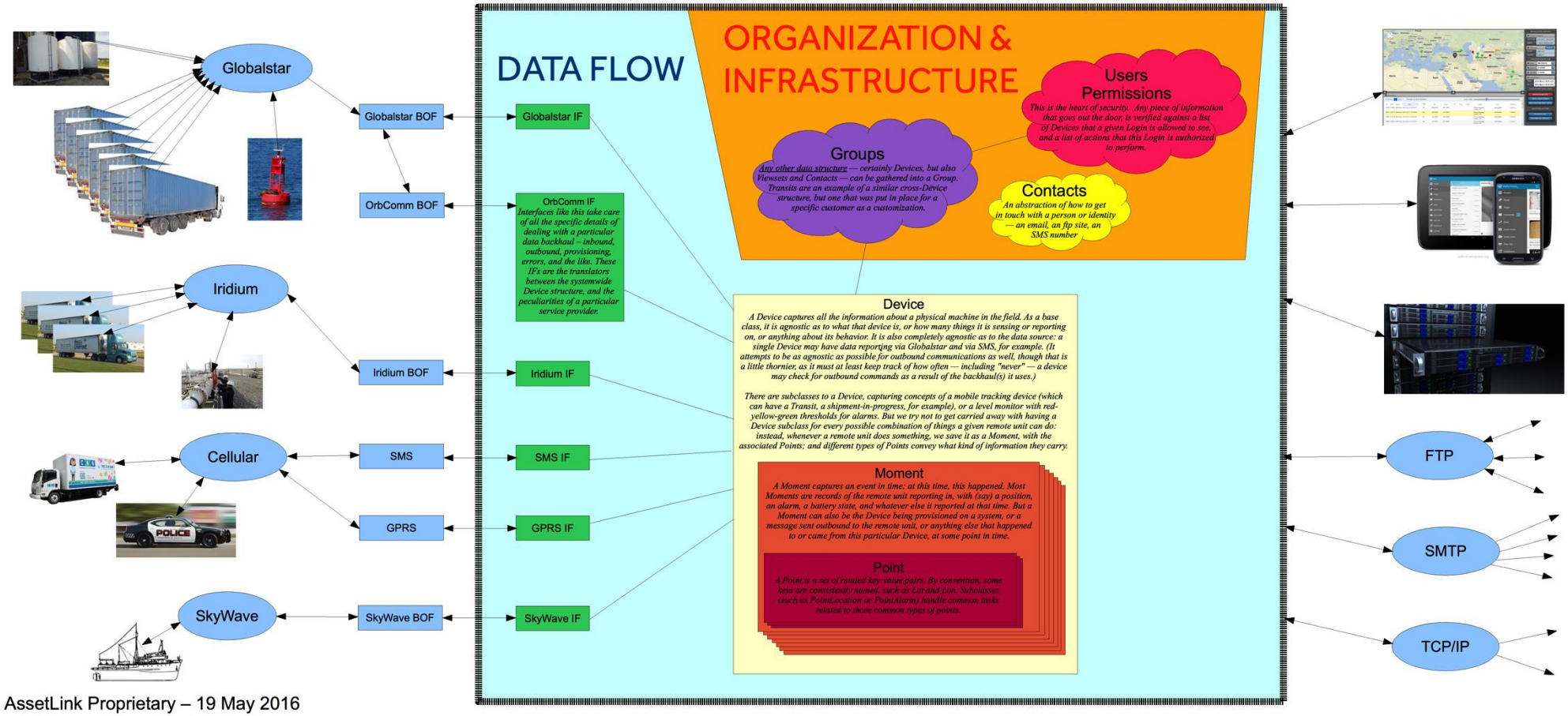
keyword in database and other pieces of infrastructure, the specific method of grouping units in the DeviceManager is shortened to "Grp".

This structure – key:values, inside Points, inside Moments, inside Devices, gathered into Grps – is the consistent structure used *for anything that occurs at some time to a device out in the world*. Hardware devices that currently exist, whose report data we already know. Hardware devices we can envision. Hardware devices we *can't* yet envision. If this hardware reports something fairly common, like a location, we record that location in a consistently-named way so that everything downstream that cares in any way about a location, can recognize that common name and use it. If this hardware reports something totally unique, it's still a key and a value in a Point in a Moment in a Device. And anyone downstream – all the way to a particular user staring at a screen – who recognizes that key, will know what that value means.

As the number of devices increases, gathering them into useful sets becomes a vital way to organize workflow. A given device may be within several different Grps, which in turn may be part of different Grp trees. A set of connected Grps typically share a "Subtype", to describe the intent of that collection of Grps. The following tree extract is a conceptual example; the actual set of Grp Subtypes can be organized and customized for each AssetLink customer.



The following pages show diagrammatically how the DeviceManager takes in, organizes, and resupplies unit information in a common form and format. Note the central text is reproduced on the second following page for easier reading.



AssetLink Proprietary – 19 May 2016

ORGANIZATION & INFRASTRUCTURE

Users Permissions

This is the heart of security. Any piece of information that goes out the door, is verified against a list of Devices that a given Login is allowed to see, and a list of actions that this Login is authorized to perform.

Groups

Any other data structure — certainly Devices, but also Viewsets and Contacts — can be gathered into a Group. Transits are an example of a similar cross-Device structure, but one that was put in place for a specific customer as a customization.

Contacts

An abstraction of how to get in touch with a person or identity — an email, an ftp site, an SMS number

Device

A Device captures all the information about a physical machine in the field. As a base class, it is agnostic as to what that device is, or how many things it is sensing or reporting on, or anything about its behavior. It is also completely agnostic as to the data source: a single Device may have data reporting via Globalstar and via SMS, for example. (It attempts to be as agnostic as possible for outbound communications as well, though that is a little thornier, as it must at least keep track of how often — including "never" — a device may check for outbound commands as a result of the backhaul(s) it uses.)

There are subclasses to a Device, capturing concepts of a mobile tracking device (which can have a Transit, a shipment-in-progress, for example), or a level monitor with red-yellow-green thresholds for alarms. But we try not to get carried away with having a Device subclass for every possible combination of things a given remote unit can do: instead, whenever a remote unit does something, we save it as a Moment, with the associated Points; and different types of Points convey what kind of information they carry.

Moment

A Moment captures an event in time: at this time, this happened. Most Moments are records of the remote unit reporting in, with (say) a position, an alarm, a battery state, and whatever else it reported at that time. But a Moment can also be the Device being provisioned on a system, or a message sent outbound to the remote unit, or anything else that happened to or came from this particular Device, at some point in time.

Point

A Point is a set of related key:value pairs. By convention, some keys are consistently named, such as Lat and Lon. Subclasses (such as PointLocation or PointAlarm) handle common tasks related to those common types of points.

Fundamental API Format

Any API request coming into the DeviceManager will look essentially like this:

```
[                more than one request can be packaged together
{                each request is a JSON object
      "file" : "where this command is defined",
      "action" : "command name",
      "filter" : "objects this command applies to",
      "data" : {
            command-specific information
      }
      "limit" : maximum number of results to send,
      "offset" : number of initial results to skip
},
other commands following the same format
]
```

The array of requests is always executed in order, and any failed request causes the remaining ones not to be attempted.

The response from the server will be an array corresponding 1:1 with the array of instructions, where each element in the array contains a "result" parameter ("success" or "error"), the "objects" that were involved, then further details about the execution of the command. For successful instructions, there will be a "data" element which in turn contains any additional information specific to that command. For successful instructions where the server suspects the result was not what the user intended, a "warnings" element is added, with strings describing what the concern is (e.g., "1 group with name 'NewGroup' already exists" when adding a group). For unsuccessful instructions, there will be an "error" element which in turn describes what went wrong. Any unsuccessful instruction will cause all remaining requests in that array to be ignored.

Optionally, the requester can include an element named 'tag', which is echoed back in the response. It can be an object, array, or simple string; whatever it is, gets echoed back. The requester can then use that tag as a convenience to understand what this response is about.

Here is an example of a request to send a “Report ASAP” command to all of the units directly in the Grp “SandyTest”:

```
[ {   "file" : "remote",
      "action" : "report",
      "filter" : "(Grp = 'SandyTest')",
      "tag" : "Sandy's units should report",
      "data" : {}
} ]
```

Let's say there were two devices in that Grp. One has transmitted recently, and one named "New unit" has never yet transmitted. A file commanding both was created, and the Moments corresponding to these commands going out also created. (N.B.: These are just examples to bring out the format; not all responses will have exactly these result objects.) The response back to the requester looks like this:

```
[ {   "result" : "success",
      "objects" : { "Devices" : [
            {
                  "deviceid" : 12345,
                  "system" : "ISB",
                  "esn" : "300398765",
                  "name" : "",
                  "CLASS" : "AP3",
                  "lasttx" : 17123711,
                  "lasttxtime" : "2016-06-28 14:04:44",
                  "affected" : {
                        "Moment" : [ 66262 ],
                        "file" : [ "/fileout/irout/200006.bin" ]
                  }
            },
            {
                  "deviceid" : 12346,
                  "system" : "ISB",
                  "esn" : "300322322",
                  "name" : "New unit",
                  "CLASS" : "AP3b",
                  "lasttxtime" : "1000-01-01 00:00:00",
                  "affected" : {
                        "Moment" : [ 66263 ],
                        "file" : [ "/fileout/irout/200007.bin" ]
                  }
            }
      ] },
      "data" : {
            "msgtype" : 11
      }
      "tag" : "Sandy's units should report"
} ]
```

... where the "affected" element for each Device shows Device-specific information relevant to (caused by) this request, and the top-level "data" element shows information relevant to this request that is not tied to a particular Device — in this case, the message type that the devices were requested to send back.

If unusual things happen in servicing the request, there are two other elements which may show up. If the "limit" of response objects is reached, an "*Object*Limited" (viz., "MomentLimited") element will be added to the "objects" structure. If a warning is generated — like, "Careful, you just added a group with the same name as one that already exists" — a "warnings" list is added to the response structure.

This document will first explore the "filter" portion of the request — the most critical part of this API that singularly encapsulates the *select* concept explored in the Introduction — and then enumerate the "actions" we will be starting with.

Filters

A given action is going to apply to some target type of object: often a Device, a Moment, or a Grp. The "filter" element thus is intended to say *which* Devices, Moments, or Grps the action should be taken on, but do so in an organized and common way.

To that end, we're going to define what we'll call the **canonical form** of the filter, which is simply a string of the following form:

(<conditional>)

... but which gets considerably more interesting as we expand the definition:

<conditional> ∈ **{**

- **TRUE**
- **FALSE**
- **(<conditional>) AND|OR (<conditional>)**
- **<Parameter> <comparator> <value>**
- **<Parameter> IS [NOT] <special>**
- **<Parameter> IN (<value>[,<value>[,...]])**

}

<comparator> ∈ **{**

=	***equals***
!=	***does not equal***
<	***less than***
<=	***less than or equals***
>	***greater than***
>=	***greater than or equals***
LIKE	***string pattern match***
NOT LIKE	***string pattern not-match***

}

<value> is either a number, and is unquoted; or is a string, and is surrounded by single or double quotes. These comparators and values are passed to the database (parameterized, of course, for safety); so the application of a comparator to a value will be interpreted according to MySQL rules.

<special> ∈ {

NULL

we may add other special values for the DeviceManager to interpret

}

The **<Parameter>**s, then, are specific to an object type.

Parameters relevant to Grps are:

Grp	*the name (string) or id (number) of a Grp*
GrpType	*the type (string) of a Grp*
GrpSubtype	*the subtype (string) of a Grp*
GrpParams	*the params (string) of a Grp*
GrpNotes	*the notes (string) of a Grp*
GrpBelow	*this Grp and all Grps below it (branching from it, recursively)*

Parameters relevant to Devices are:

Grps above, specifying the Grps that matching Devices must be a leaf of; plus

Device	*the esn (string) or id (number) of a Device*
esn	*the esn (string) of a Device*
DeviceName	*the name (string) of a Device*
lasttxtime	*the time (string, ISO8601) that the last message was received from a Device*

Parameters relevant to Moments are:

Devices above, specifying the Devices whose Moments we are interested in; plus

Moment	*the dateOriginated (string, ISO8601) or id (number) of a Moment*
dateReceived	*the dateReceived (string, ISO8601) of a Moment*

These lists can be expected to grow to expand the capabilities of the API. If there is a particular filter parameter that would be of use in your company's application, please let us know.

Non-Canonical Forms

For both historical and convenience reasons, the filter can be expressed in a few different ways. The key is that they are all ultimately reduced to the canonical form. Following are the non-canonical ways of expressing filters:

Comparators: the following substitutions are performed for <comparator>s:

- == ⇒ =
- eq ⇒ =
- <> ⇒ !=
- ne ⇒ !=
- lt ⇒ <
- le ⇒ <=
- gt ⇒ >
- ge ⇒ >=

Abbreviations: the following substitutions are performed:

- st= ⇒ Moment>=
- et= ⇒ Moment<=
- () ⇒ TRUE

URI form: a request can be sent in the URI instead of or in addition to the POST contents. It is considered the first request, and its response will be the first one in the returned array. This URI-form request should still contain at a minimum a "file=" and "action=" to define the request. From there, the filter information can be expressed in one of two ways:

- **filter=*URI_encoded_string***
- a series of **<Parameter>=<value>** pairs; in this case the **<comparator>** is always =, and there is an implicit "AND" connecting each pair. Note that unrecognized **<Parameter>**s are ignored in this case; in all other methods of presenting a filter, an unrecognized **<Parameter>** results in rejection of the request.

Web API

There is a single base URL file to call, which then dispatches as needed. This single URL is https://mapdata.assetlinkglobal.com/Portal/api.php . When a request is received over this API, if any value is not populated, it uses the following defaults:

- The default value for "file", if no other one is present, is "moment".
- The default value for "action", if no other one is present, is "get", or "getMostRecent" if the file is "moment".
- The default value for "data", if no other one is present, is null.
- The default value for "filter", if no other one is present, is the constant "TRUE".

- The default value for "limit" and "offset" is the Boolean "false", which is interpreted to mean "none". The exception is when file is "moment": then limit defaults to 1000.

One of the first things `api.php` does is a first-cut assessment of whether this user is properly logged in. `api.php` can only be called after the caller has established a session via `Portal/autologin.php`, as follows:

Login

The login structure is a single JSON object with a single element, named USER. The user element in turn contains two elements, the NAME and PASSWORD of the login to be used. Again, this can only be sent using HTTPS: do not send this (or any) information plaintext.

```
{"USER" : {
      "NAME" : "login_username",
      "PASSWORD" : "login_password"
  }
}
```

The server responds either with

```
{"USER" : {
      "NAME" : "login_username",
      "STATUS" : "VERIFIED"
   }
}
```

on success, or

```
{"USER" : {
      "NAME" : "login_username",
      "STATUS" : "UNVERIFIED"
  }
}
```

on failure. The cause of failure is intentionally not announced for security.

Logout

The logout structure is the smallest and simplest possible, to make it easiest for a session to be safely and securely closed. It consists of simply

```
{
   "LOGOUT" : true
}
```

There is no response from the server beyond the 200 HTTP response.

Returning Data

Each action defines what the relevant response is. That said, there are some general guidelines and patterns:

- **Grp** responses give all the information about the resultant Grp, plus the ids of immediate parents and immediate children of the Grp.
- **Device** responses include all items in the Device's database row, plus all immediate parent Grp ids and names.
- **Moment** responses are grouped under their owning Devices. The Device information includes all information from the Device response above, except the JSON columns ('current' and 'extra') and Grp memberships. Each Moment then contains all the Point and PointMap information beneath it.
- An action that is about **commanding a device remotely** gives traceability information on the command that went out (e.g., its filename, and perhaps the Moment corresponding to the command if a Moment is created), and may send much less information on the device itself (perhaps as little as the id and esn).

A given action may certainly deviate from these guidelines if it makes sense to do so for that action. That said, following are some examples. These would all be encapsulated inside the "objects" element of the API response.

For a request that responds with a list of Grps:

```
{ "Grps" : [
      { "grpid" : 589, "name" : "NorthAmerica",
        "type" : "Device", "subtype" : "Company",
        "params" : "", "notes" : "All units based in North America",
        "parents" : [ { "grpid" : 20, "name" : "MightyCorp" } ],
        "children" : [
            { "grpid" : 590, "name" : "Texas" },
            { "grpid" : 591, "name" : "NorthDakota" },
            { "grpid" : 592, "name" : "Canada" }
        ] },

      { "grpid" : 590, "name" : "Texas",
        "type" : "Device", "subtype" : "Company",
        "params" : null, "notes" : "All units based in Texas",
        "parents" : [ { "grpid" : 589, "name" : "NorthAmerica" } ],
        "children" : [] },

      { "grpid" : 591, "name" : "NorthDakota",
        "type" : "Device", "subtype" : "Company",
        "params" : null, "notes" : "All units based in North Dakota",
        "parents" : [ { "grpid" : 589, "name" : "NorthAmerica" } ],
        "children" : [] },

      { "grpid" : 592, "name" : "Canada",
```

```
        "type" : "Device", "subtype" : "Company",
        "params" : null, "notes" : "Units transiting Canada",
        "parents" : [ { "grpid" : 589, "name" : "NorthAmerica" } ],
        "children" : [] }
] }
```

For a request that responds with a list of Devices:

```
{ "Devices" : [
      { "deviceid" : 12612, "name" : "",
        "system" : "ISB", "esn" : "30023495122", "CLASS" : "AP3b",
        "lasttx" : 18531436, "lasttxtime" : "2016-07-26T14:42:06Z",
        "current" : { "Lat":41.464022, "Lon":-75.552142 }
        "extras" : []
        "parents" : [
            { "grpid" : 589, "name" : "NorthAmerica" },
            { "grpid" : 104, "name" : "Generator" }
        ] }
] }
```

For a request that responds with a list of Moments:

```
{ "Moments" : [
      { "deviceid" : 12612, "name" : "",
        "system" : "ISB", "esn" : "30023495122", "CLASS" : "AP3b",
        "lasttx" : 18531436, "lasttxtime" : "2016-07-26T14:42:06Z",
        "moments" : [
            { "momentid" : 18531400,
              "dateOriginated" : "2016-07-26T14:00:04Z",
              "dateReceived" : "2016-07-26T14:00:41Z",
              "points": [
                  { "Point": { "ModeChange": "4 - In Motion" } },
                  { "PointLoc": {"Lat":41.464030, "Lon":-75.553450}}
            ] },
            { "momentid" : 18531436,
              "dateOriginated" : "2016-07-26T14:42:06Z",
              "dateReceived" : "2016-07-26T14:43:01Z",
              "points": [
                  { "PointLoc": {"Lat":41.464022, "Lon":-75.552142}},
                  { "PointAlert": {"Engine":"On", "Level":2}}
            ] }
      ] }
] }
```

For a request that is an outgoing command, the response is specific to that command, but is typically the Device that was affected:

```
{ "Devices" : [
      {
            "deviceid" : 12345,
            "system" : "ISB",
            "esn" : "300398765",
```

```
            "name" : "",
            "CLASS" : "AP3i13",
            "lasttx" : 17123711,
            "lasttxtime" : "2016-06-28 14:04:44",
            "affected" : {
                  "momentid" : 66262,
                  "file" : "fileout/irout/200007.bin"
            }
      },
      {
            "deviceid" : 12346,
            "system" : "SWV",
            "esn" : "3003SKY22",
            "name" : "New unit",
            "CLASS" : "IDP800",
            "affected" : {
                  "momentid" : 66263,
                  "serversaid" : "200 OK { \"forwardid\":41414 }"
            }
      }
] }
```

Initial Actions

This API will continue to grow and evolve. New files and actions will be added as time goes on and needs develop, still adhering to the Fundamental API defined above. Shown here are the initial actions we are building from. They are grouped by file name (the "file" element in the API request). They are grouped by file name (the "file" element in the API request). Each action has an access level that the login must meet (be numerically equal to or lower than) in order to execute it. Note that "lower than" is strict: an access level of 20 means the login must be 19 or lower. Here are the current descriptions of access levels:

Access level 10: Company Root
Can perform any action

Access level 20: System Manager
Can do anything (make groups, move devices, etc.) except make new logins

Access level 30: System Monitor
Can acknowledge alarms

Access level 40: Viewer
Can view information but not change it

We are also exploring turning access into a field of flags, so there is not a single linear progression of capabilities, but a customer-defined set of capabilities a given login is allowed to possess.

File: grp

All actions in this file are about getting or setting Grp information. All filters are only those relevant to Grp.

Action: get
Return all of the Grps that match the given filter.
Access level: 105
No data
Return:
"Grps": all Grps that match the given filter

Action: addCompany
Adds a Grp of type:Device, subtype:Company. This kind of Grp is intended to organize units by their role within an organization.
Access level: 35
Data:
"name": string, *name of the new Grp to add*
"params" *(optional)*: JSON string, *for future expansion*
"notes" *(optional)*: string, *explanatory text of what this Grp is*
"parent": string or number, or array of strings/numbers, *the name or id of the Grp that is to be this new Grp's immediate parent; can be an array for multiple parents*
Return:
"Child": newly added Grp
"Parent": parent Grp(s) the child was connected to

Action: addContact
Adds a Grp of type:Contact. The filter results are respected as the set of available Grps to use as "parent" (they need not also be "Contact").
Access level: 35
Data:
"name": string, *name of the new Grp to add*
"subtype" *(optional)*: string, *'PhoneBook' for creating a set of contact information for the entire company*
"params" *(optional)*: JSON string, *for future expansion*
"notes" *(optional)*: string, *explanatory text of what this Grp is*
"parent": string or number, or array of strings/numbers, *the name or id of the Grp that is to be this new Grp's immediate parent; can be an array for multiple parents*
Return:
"Child": newly added Grp
"Parent": parent Grp(s) the child was connected to

Action: connect
Connect a Grp as a parent to either another Grp (branch) or a target object (leaf). Note that the filter is still respected as the set of available Grps to use as "parent" and branch-"child";

other targets are taken from the set of all objects this login can see. Already-existing relationships are not recreated and are considered not affected.

Access level: 35

Data:

"parent": string or number, *the name or id of the Grp that is to be connected as a new parent*

"childtype": string, *the type of object to connect as child*

"child": string or number, or array of strings/numbers, *the id or primary string identifier (viz. esn for Device) of the object(s) to add as children to the given parent; all must be of childtype.*

Return:

"Child": connected Grp(s)

"Parent": parent Grp(s) all children were connected to

Action: disconnect

Disconnect a Grp from a specified child or children. Note that to move a child from one Grp to another, a single request array should be sent, the first request being a connect, the second being a disconnect. This way if the connect has a problem, the disconnect does not occur, and the child is still accessible through the original Grp.

Access level: 35

Data:

"parent": string or number, *the name or id of the Grp that is to be disconnected as a parent*

"childtype": string, *the type of object to disconnect as child*

"child": string or number, or array of strings/numbers, *the id or primary string identifier (viz. esn for Device) of the object(s) to add as children to the given parent; all must be of childtype.*

Return:

"Child": disconnected Grp(s)

"Parent": parent Grp(s) all children were disconnected from

WARNING:

Disconnections are obeyed as sent! If a disconnect request results in a child Device being disconnected from its only parent Grp, that Device cannot be accessed by this user any longer. If a disconnect request results in a child Grp being disconnected from its only parent Grp, that Grp *and all of its unique descendants* become inaccessible via this user's login. A mistake of this type can be remedied by contacting AssetLink — the data are never actually lost — they are simply disconnected from each other. (If the *intent* was to disconnect the Grp from this user, of course, this is not a mistake.)

The most frequent use of the 'disconnect' action is to move a child from one parent Grp to another. In this case, the new parent Grp must be *connected first*, followed by the disconnect from the old parent. As described above, this should be two requests in a single request array, because if the first request (connect) has a problem, the second request (disconnect) is not executed.

File: device

All actions in this file are about getting or setting Device information. All filters are those relevant to Grp (and the Devices at their leaves), plus those relevant to Device.

Action: get
Return all of the Devices that match the given filter.
Access level: 105
No data
Return:
"Devices": all Devices that match the given filter

Action: setName
Sets the 'name' field of the Device(s) that match the given filter.
Access level: 35
Data:
"name": string, *new name to set*
Return:
"Devices": all affected Devices

File: moment

All actions in this file are about getting Moments (and their associated Points). All filters are those relevant to Grp (and the Devices at their leaves), plus those relevant to Device, plus those relevant to Moments; the return value is a list of Devices that have nonzero matching Moments, under which are the Moments for that Device, under which are the Points and PointMaps for that Moment.

Action: get
Return all of the Moments that match the given filter. This will be the most commonly used API request.
Access level: 105
No data
Return:
"Moments": all Moments that match the given filter, as described above

Action: getMostRecent
Return the Moments that match the given filter, but only one (the most recent) per Device.
Access level: 105
No data
Return:
"Moments": all Device's most recent Moments that match the given filter, as described above

File: remote

All actions in this file are about sending a command to a remote device. Not all commands will necessarily end up here; more involved remote actions (particularly application-specific ones) will get their own infrastructure and own files. But your basic commands start here.

Action: report
Send a "report ASAP" command to the remote Device(s) that match the filter.
Access level: 45
Data: ***(optional)***
"msgtype": number or string, *the message type the Device should be prompted to send. This is of course interpreted according to what kind of Device is being commanded. For AssetPack3s, the default value is 11, "Requested Message".*
Return:
"Devices": all Devices to which the command was sent, with context and traceability for the command sent

Action: changeMode
Send a "change operating mode" command to the remote Device(s) that match the filter.
Access level: 35
Data:
"mode": number or string, *the new mode as interpreted for the Device being commanded. For AssetPack3s, this is a number, 2-15.*
Return:
"Devices": all Devices to which the command was sent, with context and traceability for the command sent

Action: modifyRegularReporting
Send a "Report every X hours" command, to be applied to a specified operating mode, to the remote Device(s) that match the filter. For instance, if a Device has an operating mode representing "vehicle in motion", and it is currently reporting once per hour, this command can change that mode to report every half-hour.
Access level: 25
Data:
"mode": number or string, *the mode to be modified as interpreted for the Device being commanded. For AssetPack3s, this is a number, 2-15.*
"rate": number, *how many hours between reports; can be fractional; will be rounded as needed for specific devices. AssetPack3s, for example, operate on a 15-minute heartbeat, so this rate would be rounded to the nearest 15 minutes.*
Return:
"Devices": all Devices to which the command was sent, with context and traceability for the command sent

The remaining commands are applicable only to AssetPack3 units.

Action: modifyPiezoMotion

Send a "modify parameters related to piezo-sensor motion detection" command to the remote Device(s) that match the filter.

The piezo sensor in the AssetPack detects when the unit is shaking. The following parameters govern how to differentiate between "shaking but not going anywhere" from "shaking because the unit is traveling down the road".

Default values are noted below. Any data field not present is left unaffected on the unit. These parameters are only effective in modes that use the piezo sensor to detect motion.

Access level: 25

Data:

"pmotion_start_check_sec": number, *the unit watches how long it has been shaking during this number of seconds, to decide if it is really moving; default 300*

"pmotion_start_percent": number, *if the unit has been shaking at least this percentage (0-100) of the time over pmotion_start_check_sec seconds, it is moving; default 10; this value should be greater than pmotion_stop_percent for hysteresis*

"pmotion_stop_check_sec": number, *the unit watches how long it has (not) been shaking during this number of seconds, to decide if it is really stopped; default 300*

"pmotion_stop_percent": number, *if the unit has been shaking less than this percentage (0-100) of the time over pmotion_stop_check_sec seconds, it is stopped; default 5; this value should be less than pmotion_start_percent for hysteresis*

Return:

"Devices": all Devices to which the command was sent, with context and traceability for the command sent

The remaining commands are applicable only to Defined Capability AssetPack3 units.

Action: modifyModeSchedule

The AssetPack operates on a 15-minute heartbeat. Each operating mode has a schedule indicating what it should do at every heartbeat. This schedule is expressed as 96 two-bit values, describing each heartbeat over the course of 24 hours. Each two-bit value can be:

0 Do nothing on this heartbeat
1 Check for incoming messages ('mailbox check') on this heartbeat
2 Perform a custom behavior on this heartbeat
3 Send a position report on this heartbeat; also checks for incoming messages

What the unit does on every heartbeat — specifically, whether it communicates over the air (1 or 3) or not — is the single largest driver of power consumption on the unit. OTA communication can happen on every heartbeat in normal-power situations, but must be pared back when power needs to be conserved. The default behaviors programmed into the unit from the factory take care of this, and place normal-power modes as the even-numbered modes. Therefore, typically, modifying the schedule of even-numbered modes is a safe thing to do.

Access level: 25
Data:

"mode": number, *the mode (2-15) to be modified*

"schedule": string, *a series of 96 values, each one 0-3, indicating what behavior the unit should perform at each 15-minute heartbeat over the course of 24 hours*

"absolute": boolean, *if true, the first value in "schedule" corresponds to the first heartbeat after 00:00:00Z (i.e. between 00:00Z and 00:15Z); if false, the first value in "schedule" corresponds to the first heartbeat after this mode is entered; default false*

Return:

"Devices": all Devices to which the command was sent, with context and traceability for the command sent

Action: modifyGPSMotion

Send a "modify parameters related to GPS-sensor motion detection" command to the remote Device(s) that match the filter.

When using the GPS to detect motion, the GPS is activated every 5 minutes, and the position (if found) compared to an earlier position to determine how far the device has traveled.

Default values are noted below. Any data field not present is left unaffected on the unit.

These parameters are only effective in modes that use the GPS sensor to detect motion.

Access level: 25

Data:

"gmotion_start_radius": number, *if the unit moved more than this many meters since its last position, it is moving; default 250*

"gmotion_stop_radius": number, *if the unit moved less than this many meters since a recent position, it is stopped; default 250*

"gmotion_min_gps_sec": number, *the minimum number of seconds between GPS positions that allows a conclusion of "we've stopped"; default 200*

Return:

"Devices": all Devices to which the command was sent, with context and traceability for the command sent

Cookbook

The following page is a set of the most commonly used API commands. Oftentimes a document like this describes the manifold things a user *can* do, and the relatively small number of things a user realistically *will* do is left is a fatiguing exercise in visual sifting. To aid in getting started, then, we recognize that (conceptually) 90% of all DeviceManager interactions will be:

1) Log in
2) Get any unit reports that have come in since the last request
3) Log out

... with even #1 and #3 being optional, if you log in once and then maintain that session ID.

On a command line, this might look like (important distinctions in bold):

```
~$ curl -H "Content-Type: application/json" -X POST -c cookiefile -d '{"USER":{"NAME":"mightyuser","PASSWORD":"kapow"}}'
   https://mapdata.assetlinkglobal.com/Portal/autologin.php

~$ curl -H "Content-Type: application/json" -X POST -b cookiefile -d
   '[{"file":"moment","action":"get","filter":"(Moment>18670000}"}]' https://mapdata.assetlinkglobal.com/Portal/api.php

~$ curl -H "Content-Type: application/json" -X POST -b cookiefile -d '{"LOGOUT":true}'
   https://mapdata.assetlinkglobal.com/Portal/autologin.php
```

Note that the middle step takes advantage of the default limit of 1000 Moments per request.

Most of the remaining 10% of interactions with the DeviceManager, are other API commands as listed on the following page: provisioning units to allow their data to come through, and requests for a unit to report in as soon as possible. Once your codebase is set up to send these frequently-used requests, it will be easy to expand into the least-used requests, such as grouping, mode-changing, and naming, described earlier in this document.

When you want to…	Use…	For example,	*Notes*

Log in, because this lets you do any of the below actions	{"USER" : {
	"NAME" : "*login_username*",
	"PASSWORD" :
	"*login_password*"
	}
	}
		{"USER" : {"NAME":"ima_user",
		"PASSWORD":"Not-a-real-password"}}
			This is sent to autologin.php instead of api.php. The important part is that this interaction starts a session.

Activate a unit that's about to be deployed	[{"file" : "device", "action" : "provision", "filter" : "esn in ('*esn1*','*esn2*',...)", "data" : { "plan":"7.6kB" } }]	[{"file":"device", "action":"provision", "filter":"esn in ('300234063389400' '300234063382370', '300234063381390', '300234063371440')"}]	Drop out the "plan" to use the default billing plan.
Get the most recent reports from all units	[{"file" : "moment", "action" : "getMostRecent", }]	[{"file":"moment", "action":"getMostRecent"}]	This will be subject to the same default 1000-Moment-result limit, if not overridden. Alternately, request specific units using "filter", or set up Gr[ou]ps on the Manager and filter for Grps of interest.
Get all of the reports since last time	[{"file" : "moment", "action" : "get", "filter" : "Moment > *last_id_seen*" }]	[{"file":"moment", "action":"get", "filter":"Moment>123456"}]	Each time you call this action, note the highest Moment id in the response, and use that as the *last_id_seen* for the next request. Ids always increase.
Get all of the reports since last time, with a larger safety limit	[{"file" : "moment", "action" : "get", "filter" : "Moment > *last_id_seen*", "limit" : *max number of results desired* }]	[{"file":"moment", "action":"get", "filter":"(Moment>123456)", "limit":2500}]	With this request and the previous one, to get all the results, keep requesting this (with *last_id_seen* increasing for each response) until a response with fewer than "limit" Moments received. (This will also be indicated by the presence / absence of a "MomentLimited" element in the response.)
Tell a specific unit to send a position report ASAP	[{"file" : "remote", "action" : "report", "filter" : "esn = '*target_esn*'" }]	[{"file":"remote", "action":"report", "filter":"esn='300234063349390'"}]	Remember that 'remote' commands may incur additional air charges.
Deprovision a unit that is no longer in active service	[{"file" : "device", "action" : "deprovision", "filter" : "esn in ('*esn1*','*esn2*',...)" }]	[{"file":"device", "action":"deprovision", "filter":"esn in ('300234063349390', '300234063371440')"}]	This is primarily for units that will never be reactivated. If a unit is deprovisioned and then provisioned again, additional charges may result.
Log out, as a safety measure to guard against session stealing	{ "LOGOUT" : true }	{"LOGOUT":true}	Not necessary, but available for increased security. Sessions normally time out after 24 hours. Sent to autologin.php instead of api.php.