AssetLink DataManager Reporting API Format and Point Guide

5 June 2017

The AssetLink DataManager (DM) normalizes data it receives from units in the field. That is to say, no matter the brand or model of the remote device, or the over-the-air format it uses, or if the data contents of the device's reports has been changed by a command, the DM parses the bytes received into intelligible information. It is this intelligible information — "Motion Started" replacing both "0x5" (for the AssetLink AP3) and "Towing" (for the Skywave IDP-800), because that is what both of those mean — which is passed along via various Internet protocols to a customer's computer.

This intelligible information is formatted according to JavaScript Object Notation (JSON), a concise and human-readable format which conveys

- what information is being sent;
- the value of that information;
- how that information fits with surrounding information (structure).

This JSON encapsulation of a remote device's report may then be conveyed via any common Internet protocol: HTTP, TCP direct (push or pull), FTP (push or pull), email, and so forth.

Data Structure Syntax

We use the term 'report' to generally mean any single set of information sent from a remote device. When defining a machine-to-machine interface, however, general terms are not sufficient. As a result, we introduce the following words to more precisely indicate how report data is structured:

- at the most basic level is a set of *key:value pairs* (such as Lat:34.56, Lon:-77.02, Door:open),
- which are gathered into associated structures called *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* (abbreviated as *Grp* to avoid tripping over common keywords). 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.

When one or more reports are conveyed using the AssetLink DeviceManager JSON format, it centers on the *Moment*. Each self-contained report translates to a Moment — "at this time, the remote device had this information" — with that Moment both possessing a number of Points below, and belonging to a single Device above. The JSON-formatted result can then be broken down like this:

- At the top level, metadata about the result itself: was gathering and formatting the Moment(s) successful, or were there errors encountered
  - Under that, a list of Devices
    - Under each Device, a list of the Moments it recorded
      - Under each Moment, a list of the Points describing the information at that moment
        - Under each Point, the key:value pairs defining the numeric or textual information gathered

Here is a concrete example of such a data structure. (Note that in a typical DeviceManager API response, there can be one or more of these structures, in a JSON array; see the DeviceManagerAPI document.)

```
{                    the start of the data structure
  "result": "success",     there was no problem formatting these data
  "objects": {                    the data structure containing results
    "Moments": [                        the results for device reports, are Moments
      {                                       the start of a Device's Moments
        "deviceid": 24740,                          this Device's database identifier
        "system": "ISB",                            this is an Iridium Short Burst device
        "esn": "300234010201120",                   the serial number of this device
        "name": "Scott's Car",                      a human-helpful device name
        "lasttxtime": "2017-06-07T00:29:52Z",       the last time we got a report, UTC
        "moments": [                                this Device had the following Moments
          {
```

```
  "momentid": 20048860,                          this Moment's database identifier
  "dateOriginated": "2017-06-02T04:07:28Z",      when this happened on the unit
  "dateReported": "2017-06-02T06:25:36Z",        when the unit reported this
  "points": [                                    this Moment has these Points
   {
      "PointMsgType": {                                 this Point defines the
        "MsgType": "Report (normal)",                   type of message received,
        "num": "13"                                     both in number and name
      }
    },
    {
      "PointLoc": {                                     here is the location of
        "Lat": "38.922704458237",                       this Device at the given
        "Lon": "-77.039651870728"                       dateOriginated above
      }
    },
    {
      "Point": {                                        at this Moment, this was
        "Battery": "7.6V"                               the device's battery level
      }
    },
    {
      "Point": {                                        at this Moment, the Device
        "UnitTemperature": "16C - 23C"                  was within this tempera-
      }                                                 ture range
    }
},                                               End of this Moment
{                                                Start of the next Moment for the
                                                 same Device
  "momentid": 20051385,
  "dateOriginated": "2017-06-02T13:22:08Z",
  "dateReported": "2017-06-02T13:32:26Z",
  "points": [
   {
      "PointMsgType": {                                 This Moment was externally
        "MsgType": "Requested",                         requested (typically by
        "num": "11"                                     over-the-air command)
      }
    },
    {
      "Point": {
        "Battery": "7.6V"
      }
    },
    {
      "PointSensor": {                                  This Moment contains
        "Name": "Sensor4",                              sensor values: 3 values
        "0": "1321",                                    each, for 2 different
        "1": "1322",                                    sensors.  Sensors are by
```

```
                "2": "1321",                                    nature usage-specific,
                "enumeration": "values"                         and applications may have
              }                                                 custom parsing and inter-
            },                                                  pretation for sensor data.
            {                                                   By default, sensor Points
              "PointSensor": {                                  report one or more values
                "Name": "Sensor5",                              in milliVolts, going back-
                "0": "78",                                      ward in time. Sensor5, for
                "1": "94",                                      example, recorded 106,
                "2": "106",                                     then 94, then 78.  This is
                "enumeration": "values"                         to ensure that value 0 is
              }                                                 consistently the most
            }                                                   recent value.
          }                                                 End of second Moment
        ]                                                 End of list of Moments for this Device
      }                                                 End of information about this Device
    ]                                             End of list of Devices
  }                                             End of result-objects structure
}                                         End of data structure
```

There is a key aspect of the data structure that cannot be stressed enough: unrecognized data should be ignored. It is nearly a truism that the data structure described above will always contain data that is irrelevant (and in many cases opaque) to a given recipient. The DataManager sends this information anyway, preferring to share more than is needed than ever risk holding back some esoteric piece of data that may be important to one particular customer. That said, you can assume that any data structure your system receives from the DM, will contain a host of entries that you may not recognize. If you do not recognize them, it is almost certain that your application has no need of them.

As one example: database identifiers, as shown above, always are sent but rarely are needed. Some end applications keep track of them, particularly those using a pull API. Most applications don't, and ignore that information. This is in accordance with the general rule: do not worry about extra information, simply ignore it. (Odd as it sounds, this "freedom to ignore" is one of the great strengths of so-called self-defining interfaces like JSON and XML.)

Data Structure Semantics

The example above shows how information is organized and conveyed. At the end of the day, though, the recipient of this data structure needs to know specifically what to look for to garner useful information. (To wit, the information *not* to ignore.) For the data structure used here, there are two general kinds of data objects:

- Moments, Devices, and the top-level structure, which contain relatively fixed information like serial numbers and timestamps
- Points, which are fundamentally broad-reaching and may be application-specific

That is to say, you'll always get pretty much the same information in a Device object, but you'll get all sorts of different possibilities with each Point object.

The following tables explore each data object in turn, with the elements of that object that are most important.

TOP-LEVEL OBJECT		
Key	**Sample value**	**Description**
result	success	Whether this data structure was assembled with (error) or without problems (success)
objects	{ "Moments": [*Devices*] }	A named list of data objects (for reports, always Moments under their respective Devices)

DEVICE OBJECT (in the list under "Moments":[...])		
Key	**Sample value**	**Description**
deviceid	24740	Database identifier for this Device
system	ISB	The primary communications backhaul this Device uses. Possible values are: • GSX (Globalstar Simplex) • ISB (Iridium Short Burst Data) • SWV (Skywave) • SCL (Sierra Wireless Cellular)
esn	300234010201120	The electronic serial number of the primary backhaul modem used on this device
name	Buoy 10468	A reference for this device that is useful for the end application. Typically represents either an association with an asset ("Buoy 10468"), or a human-useful identifier ("Scott's car")
lasttxtime	2017-06-07T00:29:52Z	The date and time this device last sent a report. As with all timestamps in the DM, this is in UTC.
CLASS	AP3i15	The kind of product this is. Not often useful outside of the DM itself.
moments	[*Moments*]	A list of Moment objects

MOMENT OBJECT		
Key	**Sample value**	**Description**
momentid	20046077	Database identifier for this Moment
dateOriginated	2017-06-01T22:01:30Z	The date and time the Device said this Moment happened. Typically the time the GPS fix was achieved; or for alarms ("Door opened"), the time the alarming event occurred. If the "OldReport" key is present,

		this value should be understood as "The event occurred prior to this time." While this is reported to the second, it is up to the particular remote device to set the precision of this value.
dateReported	2017-06-01T22:08:44Z	The date and time the communications backhaul says this information was received; in other words, when the Device "got through" to convey the data in this Moment
OldReport	true	A flag to indicate if dateOriginated should be considered accurate (OldReport false or absent), or a not-later-than limit (OldReport true)
points	[*Points*]	A list of Point objects

As noted above, there are many, many possible Point objects. It should never be assumed that a particular Point object is present in a Moment. As one example, a remote device may send a location report, but be unable to get a good GPS fix. In this case the report will still come in — with other valuable information, like the unit's power state and the fact that the unit is healthy apart from its difficulty seeing GPS satellites — but will not contain a PointLoc with a location. This is not an error, it is the correct presentation of the information that was and was not received.

Points are either generic (called simply 'Point'), or have a name indicating the kind of information being conveyed (e.g. 'PointAlert', 'PointGeofence'). Each table below explores a different category of Point.

GENERIC Point OBJECT		
Key	**Sample value**	**Description**
Battery	7.6V	The current level of the unit's battery or main power rail. This value includes a units suffix, because some remote devices report battery level in Volts (7.6V), some as percent full (75%).
UnitTemperature	16C – 23C	The internal temperature reported by the unit. If a remote unit sends this as a single value, that value is reported; if it is sent as a range, this range is given. Note that the UnitTemperature is rarely the same as the outside air temperature, instead reflecting sun vs shade and internal circuitry heating.
NumSats	10	The number of GPS satellites the remote unit used to identify its position. This should be understood as identifying the GPS fix as being in one of four categories: • NumSats = 4: the unit did not get a real GPS fix; it is giving its best estimate of position short of a true fix • NumSats = 5 or 6: the unit has poor sky view • NumSats = 7 or 8: the unit has moderate sky view • NumSats >= 9: the unit has good sky view
AcqTime	50	The number of seconds the GPS was active trying to get a fix
Text	*any text string*	The unit has sent a textual response to a command
TimeSinceLastData	86400	The number of seconds since this unit last reported. The DeviceManager calculates this value when it knows a unit should be reporting regularly, to identify units that may have gone quiet.
Meta*Value*	15	Meta-information about the communication process, supplied by the communications backhaul

		provider (not by the end unit itself)
SessionStatus	2	Code from the communications backhaul provider describing how the communication concluded; only meaningful in the context of that particular backhaul

PointMsgType OBJECT[1]		
Key	**Sample value**	**Description**
MsgType	Motion Start	What kind of message this unit has sent, creating this Moment. These can be end-device- and application-specific, but for data normalization, there are a common set of MsgTypes used, defined later in this document
num	5	The raw information sent from the end unit that caused the DM to identify what type of message was sent. This is the mirror image of data normalization: it informs what the unit actually sent, before normalizing.
extra	GPS-based	Optional extra information specifying why this message was sent

PointLoc OBJECT		
Key	**Sample value**	**Description**
Lat	38.922929763794	The latitude of the remote device at this moment
Lon	-77.0396733284	The longitude of the remote device at this moment

PointAlert OBJECT		
Key	**Sample value**	**Description**
Level	2	How important this alert is, on the following qualitative scale: 1. Informative ("this is something different than the usual stream of reports") 2. Notable ("this is an interesting event, like a motion stop") 3. Important ("this is an event you're going to care about, like a geofence entry") 4. Emergency ("drop everything and look at this")
summary	*any text*	This briefly identifies what the alert actually is. Common summary keys are: • Motion, values 'Start' or 'Stop', indicating the unit has started or stopped moving respectively • Geofence, values 'Enter', 'Exit', 'Yellow', or 'Unknown', indicating the unit has arrived, left, is approaching, or otherwise changed its status, respectively, relative to a defined geofence • Panic, values 'Start' or 'Stop', indicating that a panic button has been pressed or released respectively

PointSequence OBJECT

[1] For older remote units, this may come through as a generic Point object for backward compatibility

Key	Sample value	Description
0, 1, 2...	Starting, Started, Oil temperature warning...	A sequence of values associated with numeric keys, that should be interpreted as a time-series set of related information
Name	EngineMonitor	A name capturing what all of the sequence values represent. In this example, an end unit may be connected to an engine monitor, which is reporting a series of events occurring on an engine.

PointSensor OBJECT		
Key	**Sample value**	**Description**
0, 1, 2...	1748, 1806, 1851...	A sequence of values representing a series of sensor readings. The most recent sensor value is always at key 0, with successively older readings at successive keys 1, 2, etc.
Name	Sensor4	A name capturing what sensor this is. Since this is often fundamentally installation-specific, the generic name "Sensor*N*" is often used by default within the DeviceManager.
enumeration	values	Specifies what the sequence of readings actually is. Typically it is "values", meaning the reading from the sensor itself. If the sensor is fundamentally to sense *when* an external event happened (such as an engine starting), the enumeration may be "times".
time0, time1, time2...	2017-06-01T22:06:31.406Z, 2017-06-01T21:06:31.311Z, 2017-06-01T20:06:31.202Z...	The timestamps that the sensor values were taken, if available
units	mA	The physical units of the sample values. Default is mV (milliVolts).
N	8	Number of samples taken since last report
min	1748	Minimum value seen since last report
max	1966	Maximum value seen since last report
mean	1859.4	Mean of values seen since last report
stddev	42.6	Standard deviation of values seen since last report
sum	14875.2	Sum of values seen since last report
sumsq	27658946.88	Sum of squares of values seen since last report
value_state	Green	The state of this sensor relative to its value limits: • black low: this sensor is reporting a value that is so low, it implies a bad reading that should be ignored • RED LOW: this sensor is reporting a value that is lower than its low red limit (though not lower than its low black limit) • Yellow Low: this sensor is reporting a value that is lower than its low yellow limit (though not lower than its low red limit) • Green: this sensor is reporting a value that is between its low and high yellow limits • Yellow High: this sensor is reporting a value that is higher than its high yellow limit (though not higher than its high red limit) • RED HIGH: this sensor is reporting a value that is higher than its high red limit (though not higher than its high black limit) • black high: this sensor is reporting a value that is so high, it implies a bad reading that

		should be ignored • N/A: value limit checking is not applicable for this sensor at this time
rate_state	Yellow High	The state of this sensor relative to its rate limits. The possible values are the same as for value_state, but the limits apply to how much the value of this sensor has changed over a fixed length of time.
count0	14	The number of times a digital-input sensor transitioned from 1 to 0
count1	15	The number of times a digital-input sensor transitioned from 0 to 1
timein0	8123417.151	The total amount of time, in seconds, this digital-input sensor has been 0
timein1	35632.280	The total amount of time, in seconds, this digital-input sensor has been 1

PointGeofence OBJECT		
Key	**Sample value**	**Description**
fence	312	The number of the predefined geofence that is closest to the unit
set	A	If there are multiple sets of predefined geofences, which set this Point refers to. Sets are typically letters (A, B, C), or an automatically-generated geofence internal to the unit (I [eye])
color	red	Whether the unit is inside (red), outside (green), outside-but-nearby (yellow), or some other relationship (unknown) with respect to this geofence
Geofence	Inside Fence 312	A textual description of the state of the unit with respect to its closest geofence

Message Types

In the interest of data normalization, as noted at the beginning, the original information sent by the remote device might be translated into a common terminology. This is most prevalent in the parsing of message type (MsgType) information — the answer to *why* a message was sent, not just the contents of the message itself. Following are common MsgTypes. Not all units can or will send all MsgTypes; and many MsgTypes are partially redundant, differing in details such as what kind of alarm sensor might be hooked up to a remote unit (in turn translating to a MsgType of "Panic Started" vs. "Door Open"). Similarly, not all MsgTypes possible for the DM to send are enumerated here: there are many that are application-specific and would only serve to obscure the useful items in this list.

Report

This is the most common type of message: the unit is reporting in, on the schedule expected of it. Frequently following this word "Report" is parenthetic information that may be of further use; different pieces of parenthetic information can be strung together, separated by commas: for instance, a MsgType may be "Report (normal, power save)".

(normal)	the unit is in an unexceptional state, generally meaning it is not moving
(motion)	the unit is reporting while in motion
(alerting)	the unit is in a non-normal state (such as, it should have a door sensor connected, and it is not)
(input low)	the unit's primary digital input is 0
(input high)	the unit's primary digital input is 1
(power save)	the unit is in a power-conserving state

Mode change to *name or number*

The unit has changed its operating mode and behavior. If the DeviceManager knows the name or description of the mode changed to, it can use it, otherwise it may show a mode number unique to the remote unit model.

Mode change (for power) to *name or number*

Same as the previous, but the remote unit has let us know that the reason for the change was because the power condition had changed.

Motion Started

The remote unit has started moving. This can be detected at the remote unit by vibration, GPS position changing, or other means; whatever message that remote unit model uses to indicate "no longer stationary", gets this message type.

Motion Stopped

The remote unit has stopped moving. The detection method options are the same as for Motion Started, and similarly, a message whose purpose is to indicate the unit is now stationary gets this message type.

Geofence Enter / Exit / Change

(The message type is one of these three, which are all of a piece.) The unit has changed its current geofence state. Any change can be indicated by "Geofence Change", with the change itself indicated by the PointGeofence within the Moment for

this message. If a remote unit has explicit and separate messages for Geofence Enter and Geofence Exit, they are respected and conveyed as separate Geofence Enter and Geofence Exit MsgTypes.

Panic Started

The remote unit has announced that some explicitly alerting event has occurred: a panic button has been pressed, an alarm has been triggered, an input signal which is normally low has gone high.
Depending on the application, this message type can also be termed "Panic Button Pressed", "Security Loop Disconnected", "Engine Started", or "Door Opened".

Panic Follow-up

For "Panic Started" messages which really do imply a panic situation, the remote unit may send off a quick message to state that the panic has occurred, followed by a different message to fill in vital information that takes time to acquire (most usually, the GPS position). This is the latter message.
Depending on the application, this message type can also be termed "Panic Button Location Update" or "Security Loop Location Update".

Mild Panic

Something important has been detected by the unit, though its importance is not as high as a full panic. For example, a SPOT personal locator has a "911" button (which would be translated to Panic), and a "Help" button (which would be translated to Mild Panic).

Mild Panic Follow-up

Same as Panic Follow-up, for the Mild Panic condition.

Panic Stopped

The panic condition that had previously been reported, is now cleared.
Depending on the application, this message type can also be termed "Security Loop Connected".

Requested

This message was externally requested. Typically the request comes via an over-the-air command out to the unit. Some end products may have local "Send a manually requested message" button, and those messages would show this MsgType as well.

Text

The end unit has sent textual information, likely in response to a command. This text is product-specific and conveyed to the end user through this MsgType.

Exception

The end unit has sent a diagnostic or status message, separate from normal reporting. A parenthetic explanation of what the exception is follows in the MsgType (such as "Exception (Unit activated)"), and the 'extra' field in PointMsgType often contains quantitative information related to the exception.

Exception messages include but are not limited to:

(Unit activated):	this unit was just turned on in the field. 'extra' may contain a timestamp of when this happened.
(Manufacturing Report):	a special message sent out during manufacturing tests
(Power Recovery checkin):	this unit is in a very low power mode, and is sending a special message to show it is still alive but trying to regain power
(exceeding hot limit):	this unit has exceeded its upper operational temperature limit. 'extra' may contain the unit temperature that triggered this exception.
(exceeding cold limit):	this unit has exceeded its lower operational temperature limit. 'extra' may contain the unit temperature that triggered this exception.
(back in temperature limits):	this unit is once again within the safe temperature range. 'extra' may contain the unit temperature that triggered this (resolution of an) exception.

As mentioned several times, this is not an exhaustive list of all message types for all end products in all applications. Such product- and application-specific messages are, by nature, both familiar and particular to an end customer's situation, and are used entirely because they are useful to the customer and ease their operations.