

**Xuvasi Ltd**

**Asset Interoperability Protocol:**

**Extending HyperCat towards a real IoT!**

**V1.0**

First Issued: 01 February 2015

This Issue: 30 July 2015

Overview:

This document provides an overview of a viable Asset Interoperability Protocol. The protocol has been developed from the HyperCat *catalogue* baseline. The protocol as presented, however, goes further than simply providing a catalogue. Instead, it extends the HyperCat structure to allow both per asset and bulk asset reporting of state; each form of which can support multiple sensors per asset.

We envisage this protocol being employed by any of:

1. A device manufacturer who wants each device to be able to report state to a central platform;
2. A gateway manufacturer who wants to report the state of any and all devices connected to their gateway to that same central platform;
3. A niche service provider who wants their system to be able to participate in a wider functional domain;
4. A software developer who, in order to work in Smart Cities or IoT demonstrators, must comply with the requirement to use HyperCat or, at least, HyperCat inspired/compliant interoperability;
5. An app developer, or owner, who wants the output from every instance of their app to be accessible from a central platform, and/or
6. An innovative service business who wants to exploit the value add arising from the availability of central platforms depicting the state of numerous ‘assets’.

As there are many extant protocols for Building Management Systems and similar legacy[[1]](#footnote-1) systems, not to mention the daily increment to the ‘standard’ and protocol activities in relation to the Internet of Things (IoT) and Smart Cities (SC), we fully anticipate that this protocol will evolve beyond the current version 1.0.

As we are fully aware that this version of the protocol contains significant overhead, which must be reduced if networks are to be able to support the forecast billions of devices entering the IoT, such evolution is accepted as a ‘good thing’. We are also keenly aware that, in the absence of de facto standards, no one emerging standard is going to be adopted by any device manufacturer or service provider. Instead, each such manufacturer or provider will tend to support up to three emerging standards in order to ‘hedge their bets’.

To that end, our vision beyond version 1.0 is towards a factory adapter model that allows any downstream protocol to be converted to any upstream protocol. This is, patently, a non-trivial exercise and one on which we are already working. That said, we are sure that you have your own preferred formats and structures and, therefore, invite you to participate in our internal conversation via **info@xuvasi.com**.

Notes:

With regard to this version of the protocol:

* Every value is a string:
  + This is to provide for common processing of a common structure.
  + We envisage this remaining the case.
* Every value is in ‘real world’ units:
  + We do not know what a fluctuation in voltage or resistance means in your context and therefore request that you provide calculated outputs
  + If you really do need to send electrical values, you will have to advise us of the calculation necessary to present that in ‘real world’ form
* By design, this is currently a read-only protocol:
  + For now
  + Do talk to us if you really need a control channel
* The protocol operates over HTTP using the POST verb:
  + If you need to use a GET, talk to us
  + If you need to use another network protocol, talk to us
* The protocol uses HTTP Basic Authentication:
  + If you need something different, talk to us
* The protocol POSTs JSON data in the request body:
  + If you need something different, talk to us

The Protocol:

As noted above, the protocol employs an HTTP Basic Authenticated POST to the a standard URL endpoint. Each POST carries a JSON payload in the request body. That JSON payload must comply with the data structure definition, below.

The data structure supports either individual or batched device reporting, with each device having unlimited opportunity to report sensor readings associated with that device. It is your responsibility to implement a server-side protocol handler.

This is currently a uni-directional (ie: data provider to server) protocol. Aggregated data, as sent to the server, is then available via the server interface. A return path protocol, allowing aggregated data to be returned to the data provider, and a control channel are under development.

The Data Structure:

As noted previously, the data structure is formed of an extension to the HyperCat catalogue standard. As such, and per the HyperCat specification, the basic structure has two components: **item-metadata** and **items**.

This is illustrated in Figure 1, below.

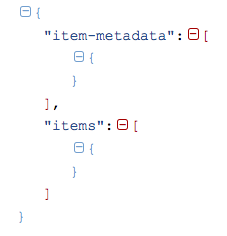


Figure 1: Basic Structure

The item-metadata node correlates directly with the HyperCat item-metadata node, albeit with a different content type (assetalert). This allows for differentiation between this data structure and more traditional catalogue structures arising from HyperCat.

Figure 2, below, illustrates a populated item-metadata component as emitted by a Building Management System (BMS) emulator running on an Intel Galileo board.



Figure 2: item-metadata Component in Structure

While the ‘supportsSearch’ element is not currently used by our server-side implementation, as all data is by definition searchable, the element remains for compliance with the HyperCat specification.

The items component, on the other hand, is divergent from the HyperCat standard due to its extension to support single or multiple device state reports, each of which may contain one or more sensor readings.

Figure 3, below, illustrates the basic structure of the items component of the data structure.



Figure 3: Base items Component in Structure

The key points to note from Figure 3 are:

1. The ‘site-ref’ element uses ‘@’ notation as part of the site reference. This is to allow convergence of data structures in this protocol with semantic models arising from Project Haystack. Upper-ontology mapping, within a server-side platform, allows for site-ref values to be translated between such ontologies as required.
2. The ‘asset-id’ element contains the unique reference, within each instance of a server platform, for the device that is reporting. This should be set when a device is registered with the server platform as this is the primary handle through which the device is managed and against which data is recorded. This is correlated with the ‘href’ element that provides single-step access to the device record within the server platform.
3. The ‘asset-type’ element contains a shorthand reference to the device type. While there is an enumeration of standard references, this can be extended on request.
4. The ‘asset-status’ element is shown set to ‘NORMAL’. It can, of course, be set to other values such as ERROR, WARNING, FAILURE, SHUTDOWN, etc.

The ‘asset-details’ and ‘sensors’ components are discussed in more detail below.

It should also be noted that Figure 3 illustrates a data structure containing only one device report. As the items component is a JSON array, there is the potential to have many device reports contained in a single message.

Figure 4, below, illustrates the items.asset-details component.



Figure 4: items.item.asset-details Component in Structure

The items.item.asset-details component is designed to contain elements that describe the device reporting state. The elements shown in Figure 4 are the minimum supported set. Additional elements may be added if required.

We note that the repetition of this detail with each report creates an unnecessary overhead and concur that a more elegant solution would be to simply maintain this information on the server platform: referenced by the asset-id element described previously.

That is, however, a future consideration as, at this time, the objective is to facilitate both use of the asset interoperability protocol and debugging of implementations towards such use.

Figure 5, below, illustrates the items.item.sensors component of the data structure. This is where the individual sensor readings of each reporting device, either singly or in batch, are recorded for transfer to any server platform.



Figure 5: items.item.sensors Component in Structure

As Figure 5 illustrates, the items.item.sensors component is an array of standard sensor components, each of which represents an individual sensor and it’s state. There is no limit to the number of sensors that may be reported: other than, perhaps, common sense.

Within each sensor component, the following:

1. The ‘type’ element describes the sensor. It is preferred that this is formed using underscores to separate each word in the sensor type description.
2. The ‘details’ component supplies:
   1. The ‘value’ element. This is a string representing the ‘real world’ value reported by the sensor. See Notes, previously.
   2. The ‘units’ element. This is a string stating the ‘real world’ units that the value is being reported in. See Notes, previously.

We have previously evaluated the performance of the asset interoperability protocol through use of an in-house BMS emulator. In that evaluation, we ran reporting from 500 devices with up to 35 sensors for each device. Even with the acknowledged overhead in the current version of the protocol, performance was well within acceptable levels.

Figure 6, below, illustrates a fully populated example of the data structure as used by the asset interoperability protocol. A copy of this message, for use in development of your own data structures, is provided with this document.

We recommend that you use <http://json.parser.online.fr> to ensure that your data structure implementations are fully compliant with the JSON specification and, of course, with this data structure.

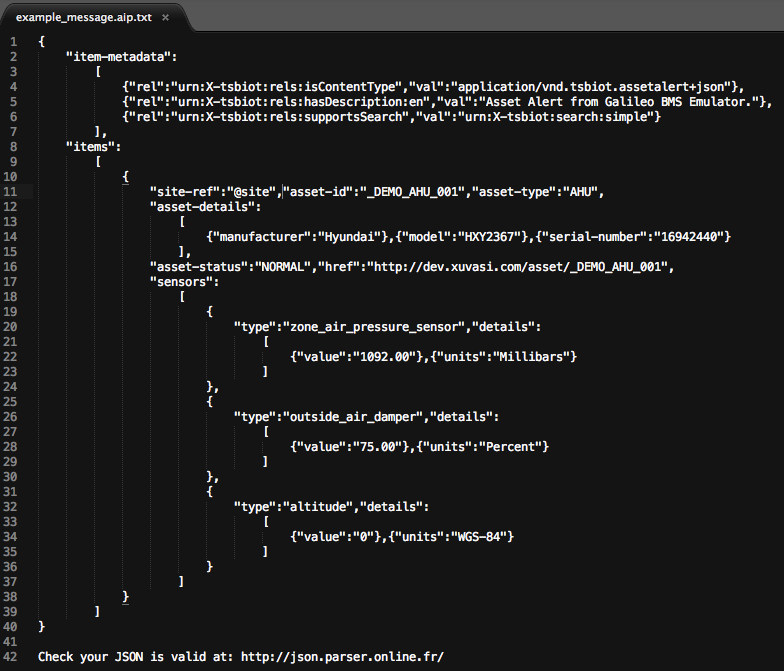


Figure 6: Example Message using Structure

Contact info@xuvasi.com with any queries, comments, or requests for further information.

\*\* ENDS \*\*

1. In the sense of ‘deployed’, not ‘old’! [↑](#footnote-ref-1)