EdgeX Binary Encoder Requirements

At the last TSC the requirement was identified to support binary data (samples) in EdgeX. This document is to outline the technical requirement for such an encoder. Architecturally the intent is that multiple encoders could be supported, with the functionality internally abstracted as much as possible so as to allow alternate implementations. However the desire is to deliver a single implementation as part of the "standard" open source project.

Initially CBOR (http://cbor.io) and protobuf (https://developers.google.com/protocol-buffers/) have been suggested as implementations, although other protocols such as the Thrift binary encoding (https://thrift.apache.org/) are potential candidates.

Requirements

This is an initial set of requirements in no particular order.

1. Preferably conforming to a standard.
2. Should have mappings in C and Go
3. Should support end to end binary data encryption and/or signing for sub contents.
4. Should preferably not rely on generated code (not a schema-less encoding).
5. Should be as lightweight and efficient as possible.
6. Can easily be converted to/from a JSON representation.

Notes

1: As an open source project the preference is to use open standards wherever possible, with adherence to IoT related standards if possible.

2: Going forward C and Go device SDK's are to be supported, so support for these languages is mandatory for any binary encoder. Java support is also highly desirable as there may be requirements to add binary data support to this existing SDK.

3: This is a nice to have rather than a mandatory requirement, as encryption of binary data can always be done as external to EdgeX. However if supported by a protocol this adds a standard mechanism for the end to end encryption of data.

4: The preference is for a protocol that includes schema as opposed to one that does not. This gives additional interface resilience in the face of change, for example supporting the addition of new data elements to an encoding without breaking existing implementations. This is one of the major reasons why IoT type systems have favoured encodings such as JSON as opposed to more rigid data type definitions (e.g. as seen in CORBA).

5: There are several facets to this, including:

- Library size
- Size of any generated code
- Speed of encode/decode
- Size of binary encapsulation

6: As JSON encoding is used internally by EdgeX, and provides a human readable format, easy conversion to/from JSON is a plus.

Some of these requirements are mutually antagonistic, specifically the desire to have an encoding that carries schema but also minimum encoding size, as clearly an encoding with schema must also carry the schema (meta data) as well as the data. However for bulk data transfer, typically the size of the data is the over whelming contributor to encoding size, with good data compression being far more significant in reducing size. Also WRT to data encoding speed, this is typically insignificant when compared to the latency of sending the data across some sort of network connection (RPC end to end latencies are typically over 95% transport).