SCHEMA DESIGN

May 2017

Dave Kemp -- NSA, Cybersecurity Architecture and Strategies

Status

- Updates since Sept 2016 Forum face-to-face:
  - **Implementation**: Python classes **-->** JAEN document
    - Test-driven design, full suite of unit tests using example data
    - Encoder methods
    - JSON-minified serialization format
    - Flattened API supports dicts and arrays
  - **Targets**: CybOX 2.1 **-->** streamlined STIX Cyber Observables
    - Attribute {“type”: “ipv4”, “value”: “1.2.3.4”}
      --> Property {“ipv4”: “1.2.3.4”}
  - **Response**: action **-->** separate datatype
  - **Modifiers**: refactored

Purpose: Interoperability

- Schema serves two independent purposes:
  - Message Definition and Validation
    - Unambiguous specification
    - Required even without serialization
  - Message Serialization
    - API / JSON – standard, human readable, ensures interoperability
    - Optional Transformations: (JSON-Minified, Binary) – allows performance optimization if supported by both sender and receiver

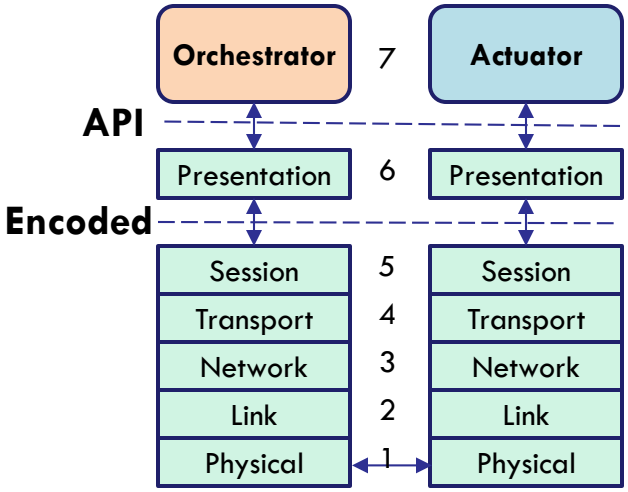
Message Definition and Validation

- Database: SQL vs NoSQL
  - NoSQL databases are built to allow the insertion of data without a predefined schema. Developers have typically had to add application-side code to enforce data quality controls, such as mandating the presence of specific fields, data types or permissible values.

    --- "*NoSQL Databases Explained*", https://www.mongodb.com/nosql-explained
  - SQL query is just text. NoSQL models require coding in some language.
    - "The more the DB engine knows about the Data Model, the more it can do for you"
  - Best practice: use right tool for the job – SQL *and* NoSQL
- Protocol data models: Schema vs. Property tables
  - No schema: Validation requires program coding, updates require re-coding
  - Schema is just text: Codec/validation engine doesn't change when protocol is updated

Message Serialization

- Serialization is a network layer six function:
  - "The Presentation layer provides independence from data representation by translating between **application** and **network** formats." -- Wikipedia
  - **application**: "state of an object instance" in a program
- Codec is application-independent
  - Implemented as a library called by applications (as with compression, signature, encryption)
  - Internet Text Messages (RFC 822) / MIME Internet Message Bodies (RFC 2045) define e-mail application payload
  - ZLIB (RFC 1950), S-MIME (RFC 5751) define Presentation-Layer transformations of payload
  - Existence of transformations **does not** create e-mail application incompatibility
- There is only one OpenC2 API
  - Define schema-based transformations in separate document



Bandwidth Optimization

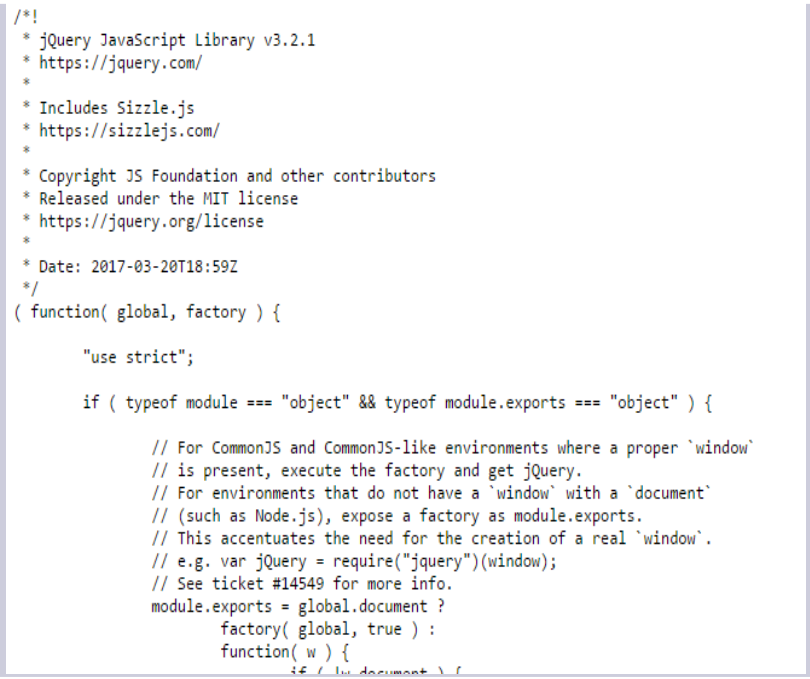
Example	JSON-Verbose*	JSON-Minified	Compressed**
T1_mitigate	77	30	164
T2_contain	170	88	216
T3_deny	390	190	351
T6_update	359	222	335

Message size in bytes *No whitespace **WinRAR compression

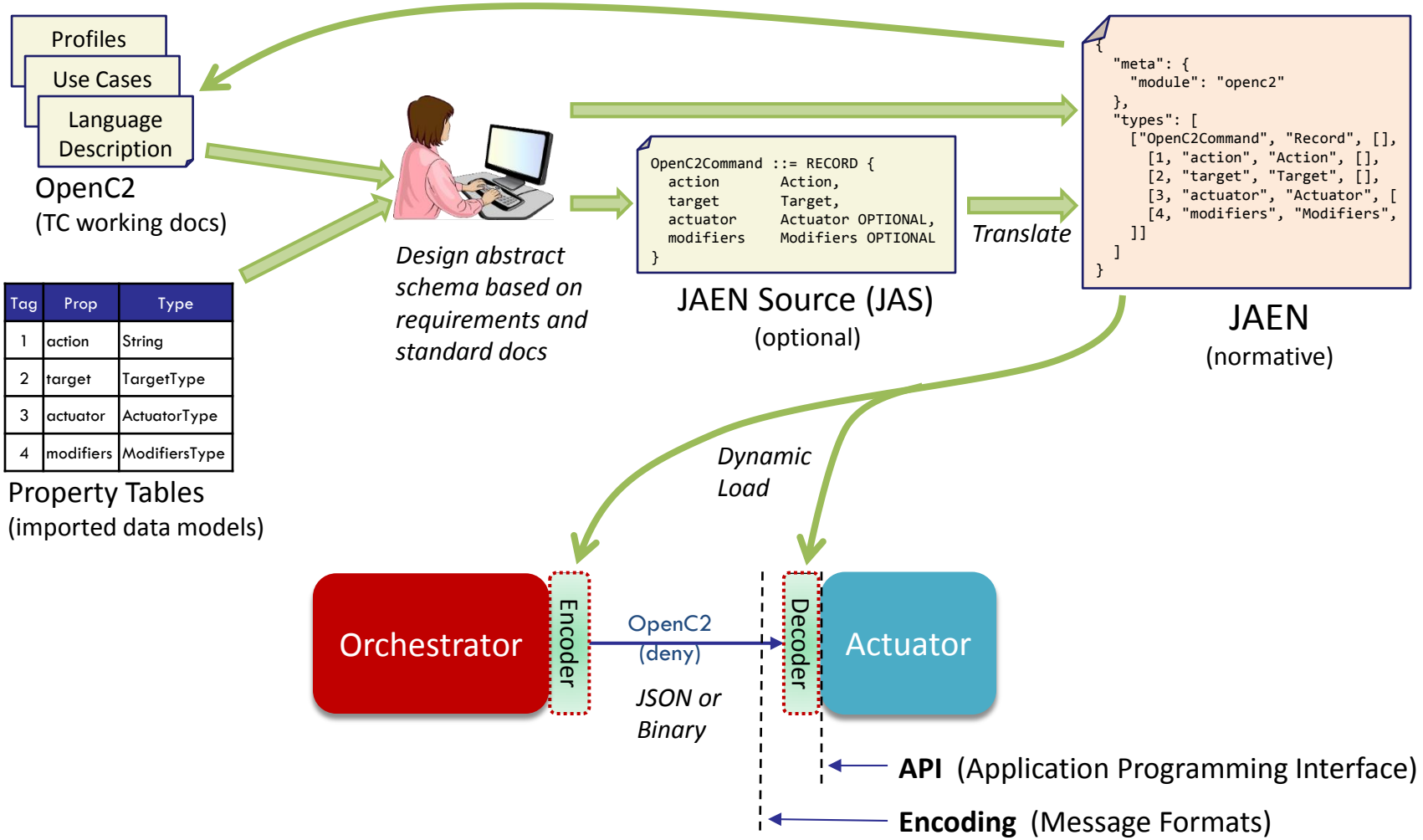
https://github.com/OpenC2-org/jaen/tree/master/examples

Schema-based optimization is more effective than compression for small messages

Web Optimization – a standard practice

Verbose – human readable	Minified – machine optimized
	
https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.js -- 262 KB	https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js -- 85 KB

Schema Design and Use



Auto-generated Property Tables

JAEN

```
["OpenC2Command", "Record", [], "", [
   [1, "action", "Action", [], ""],
   [2, "target", "Target", [], ""],
   [3, "actuator", "Actuator", ["?"], ""],
   [4, "modifiers", "Modifiers", ["?"], ""]]],

 ["OpenC2Response", "Record", [], "", [
   [1, "status", "status-code", [], "Adapted from HTTP Status Codes, RFC 7231"],
   [2, "statusText", "String", ["?"], "Status description"],
   [3, "response_src", "device-id", ["?"], "ID of the responder/actuator"],
   [4, "command_id", "command-id", ["?"], "Command unique identifier, from \"command_id\" modifier"],
   [5, "results", "String", ["?"], "Results of executing the command"]]],
```

Property Tables

	A	B	C	D	E	F
10						
11	**Type Name:**	OpenC2Command				
12	**Type:**	Record				
13						
14		**Id**	**Name**	**Type**	**Description**	
15		1	action (required)	Action (vocab)		
16		2	target (required)	Target (Choice)		
17		3	actuator (optional)	Actuator (Choice)		
18		4	modifiers (optional)	Modifiers (Map)		
19						
20						
21	**Type Name:**	OpenC2Response				
22	**Type:**	Record				
23						
24		**Id**	**Name**	**Type**	**Description**	
25		1	status (required)	status-code (vocab)	Adapted from HTTP Status Codes, RFC 7231	
26		2	statusText (optional)	String	Status description	
27		3	response_src (optional)	device-id (String)	ID of the responder/actuator	
28		4	command_id (optional)	command-id (String)	Command unique identifier, from "command_id" modifier	
29		5	results (optional)	String	Results of executing the command	
30						

Meta | Types | Vocab

Auto-generated Property Tables (cont.)


```
["status-code", "Enumerated", [], "Adapted from HTTP Status Codes, RFC 7231", [
   [102, "Processing", "1xx: Informational"],
   [200, "OK", "2xx: Success"],
   [400, "Bad_Request", "3xx: Redirection"],
   [401, "Unauthorized", "4xx: Client Error"],
   [403, "Forbidden", ""],
   [500, "Internal_Server_Error", "5xx: Server Error"],
   [501, "Not_Implemented", ""]]],

 ["encryption-algo", "Enumerated", [], "Replace with an IANA algorithm registry?  Which one?", [
   [1, "AES128-ECB", "Advanced Encryption Standard (AES) with Electronic Codebook (ECB) mode, NIST SP
800-38A"],
```

JAEN

	A	B	C	D	E
93	**Vocabulary:**	status-code			
94	**Description:**	Adapted from HTTP Status Codes, RFC 7231			
95					
96		**Id**	**Value**	**Description**	
97		102	Processing	1xx: Informational	
98		200	OK	2xx: Success	
99		400	Bad_Request	3xx: Redirection	
100		401	Unauthorized	4xx: Client Error	
101		403	Forbidden		
102		500	Internal_Server_Error	5xx: Server Error	
103		501	Not_Implemented		
104					
105	**Vocabulary:**	encryption-algo			
106	**Description:**	Replace with an IANA algorithm registry? Which one?			
107					
108		**Id**	**Value**	**Description**	
109		1	AES128-ECB	Advanced Encryption Standard (AES) with Electronic Codebook (ECB) mode, NIST SP 800-38A	
110		2	AES128-CBC	AES with Cipher Block Chaining (CBC) mode, NIST SP 800-38A	
111		3	AES128-CFB	AES with Cipher Feedback (CFB) mode, NIST SP 800-38A	
112		4	AES128-OFB	AES with Output Feedback (OFB) mode, NIST SP 800-38A	
113		5	AES128-CTR	AES with Counter (CTR) mode, NIST SP 800-38A	
114		6	AES128-XTS	AES with XEX Tweakable Block Cipher with Ciphertext Stealing (XTS) mode, NIST SP 800-38E	
115		7	AES128-GCM	AES with Galois Counter (GCM) mode, NIST SP 800-38D	

Meta | Types | Vocab

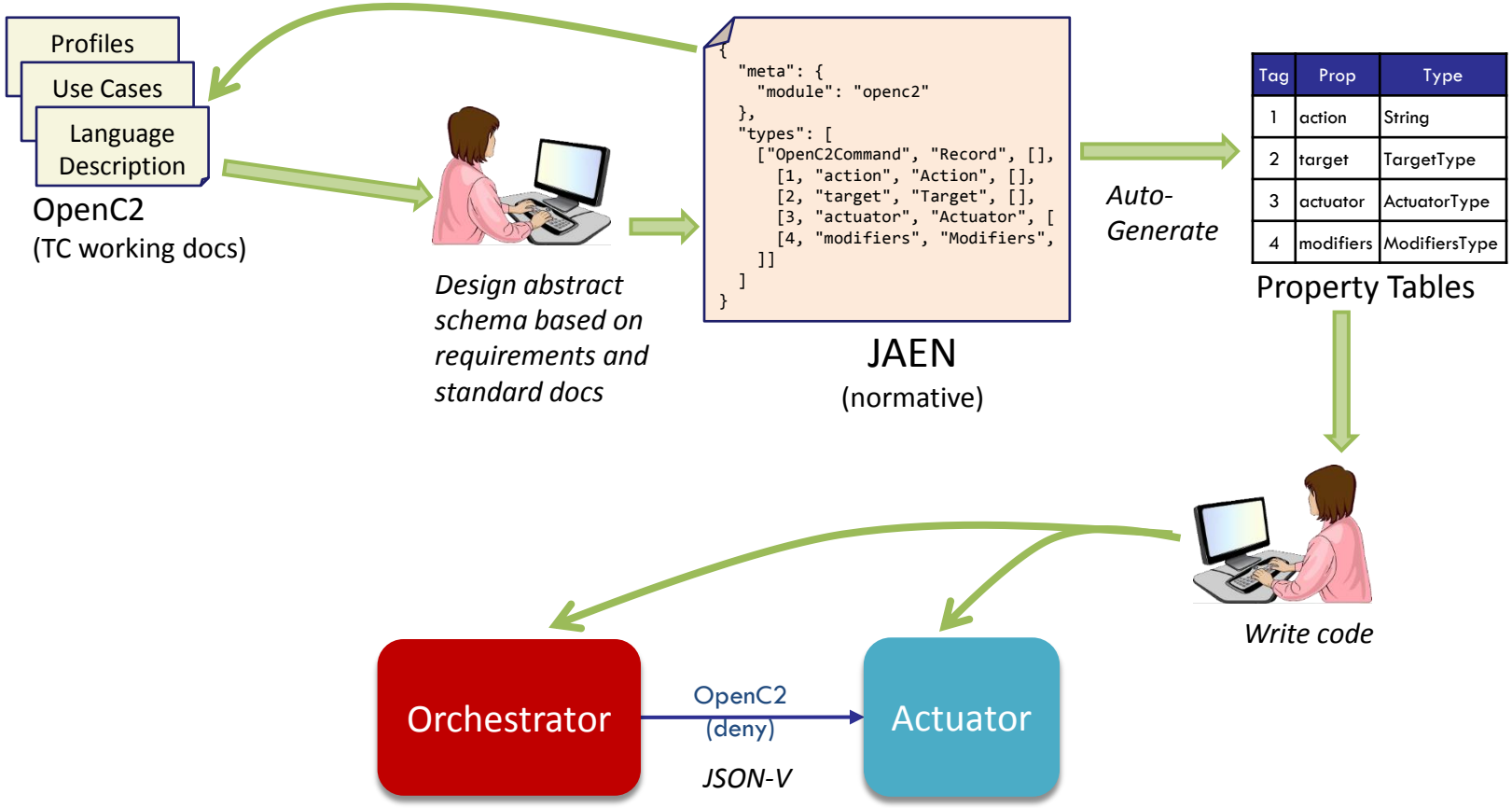
Property Tables

Auto-generated Property Tables (cont.)

- JAEN is not a mysterious foreign language
  - Machine-readable (JSON)
  - Constrained content
    - Represents a valid schema (verifiable)
    - Structured text (but description field is free-form)
  - Directly representable in property table format
    - By design

MRPT

Schema Design and Use (hand-coding)



OpenC2 Message Definition

- STIX: Standard contains property tables and JSON examples
  - Property tables are *normative* but *not testable*
  - Member-developed JSON Schemas are testable but are not part of the STIX standard
  - No path to efficient message encoding for production systems

- OpenC2: Standard contains schema and examples
  - Abstract schema is *normative* and *testable*
  - Property tables are auto-generated from normative schema
  - Member-developed concrete schemas (JSON and Binary) are derived from abstract schema and are testable
    - Goal: auto-generation of concrete schemas from abstract schema
  - Examples have been validated against normative schema

Who uses schemas?

Data Serialization:

- Apache Avro
- JAEN

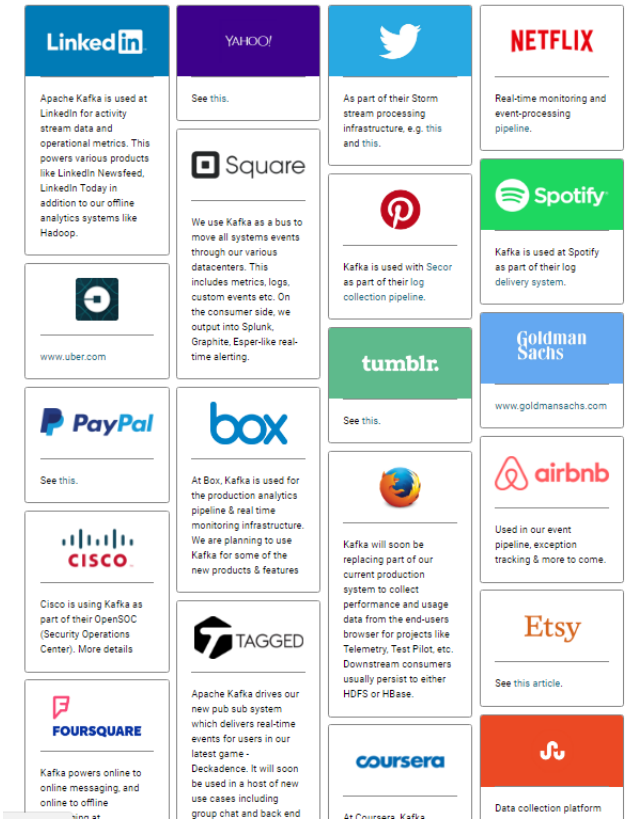
Messaging:

- Apache Kafka
- OpenDXL

"Though Kafka itself is serialization-agnostic, LinkedIn has standardized on the use of Apache Avro as the schema and serialization language for all of its activity data messages as well as data in Hadoop and most other data systems. In uncompressed form our Avro data was roughly 7 times smaller than XML messages in the activity logging system we replaced."

"This pipeline currently runs in production at LinkedIn and handles more than 10 billion message writes each day with a sustained peak of over 172,000 messages per second."

"*Building LinkedIn's Real-time Activity Data Pipeline*"
http://sites.computer.org/debull/A12june/pipeline.pdf



https://kafka.apache.org/powered-by

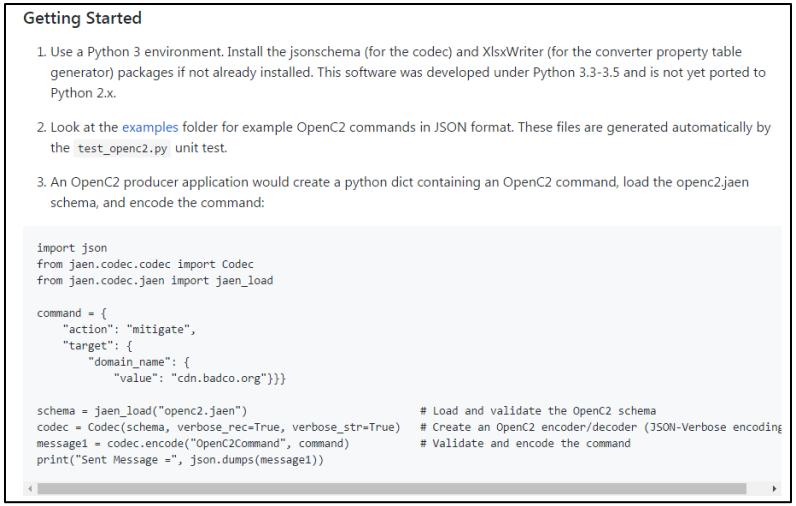
OpenC2 Design Choices

> "Avro specifies two serialization encodings: binary and JSON. Most applications will use the binary encoding, as it is smaller and faster. But, for debugging and web-based applications, the JSON encoding may sometimes be appropriate."
>
> --- https://avro.apache.org/docs/current/spec.html

- Avro is widely deployed, why create JAEN?
  - Direct correspondence with STIX property tables
  - Avro schemas are mandatory; JAEN interoperates with schema-less implementations
  - Multiple JSON encodings
    - JSON-minified is an efficient text-based alternative to binary
  - More flexible reverse-engineered abstract schemas?
    - Difficulty of specifying some existing JSON datatypes (e.g., ATVs) in Avro is unknown
- Way forward:
  - Short term – add Avro capabilities to JAEN, harmonize nomenclature
    - Namespaces, binary encoding, fixed datatypes, …
  - Mid term – Switch OpenC2 normative schema to Avro or CDDL if feasible
    - Would require property table generation

Getting Started

- Install JAEN software
  - https://github.com/OpenC2-org/jaen
- Run example_app
- View and run unit test files
  - Comprehensive examples for using and testing the JAEN codec
- If schema is modified:
  - Run jaen_convert to generate derived schemas
  - Re-run unit tests to generate example messages



Getting Started

1. Use a Python 3 environment. Install the jsonschema (for the codec) and XlsxWriter (for the converter property table generator) packages if not already installed. This software was developed under Python 3.3-3.5 and is not yet ported to Python 2.x.
2. Look at the examples folder for example OpenC2 commands in JSON format. These files are generated automatically by the `test_openc2.py` unit test.
3. An OpenC2 producer application would create a python dict containing an OpenC2 command, load the openc2.jaen schema, and encode the command:

```
import json
from jaen.codec.codec import Codec
from jaen.codec.jaen import jaen_load

command = {
    "action": "mitigate",
    "target": {
        "domain_name": {
            "value": "cdn.badco.org"}}}

schema = jaen_load("openc2.jaen")                          # Load and validate the OpenC2 schema
codec = Codec(schema, verbose_rec=True, verbose_str=True)  # Create an OpenC2 encoder/decoder (JSON-Verbose encoding
message1 = codec.encode("OpenC2Command", command)          # Validate and encode the command
print("Sent Message =", json.dumps(message1))
```

FAQ

Q: If targets have no “type” property, how does one find their type?

A: Mapping types (Python dicts, JSON objects) are collections of key: value pairs. When an object has only one item, the key is the type of that item.

Attribute:

```
target = {"type":"ipv4", "value":"1.2.3.4"}
type = target["type"]
```

Property:

```
target = {"ipv4": "1.2.3.4"}
for key in target: type = key
```

-- or --

```
type = next(iter(target))
```

BACKUP