OpenC2

SCHEMA DESIGN

Sept 2016

Dave Kemp -- NSA, IA Architectures & Mission Applications

Schema

A schema is:

- *Generic:* a structured framework or plan
- *Database:* the structure of a database system, described in a formal language that defines the tables, the fields in each table, and the relationships between fields and tables
- *XML/JSON:* a description of the elements in a document that can be used to validate each piece of content

A schema can be:

- abstract or concrete
- formal (written in a syntax definition language) or informal

Abstract Syntax*

An Abstract Syntax Language is:

- A formal language for specifying the logical structure of data that is to be exchanged between two endpoints
  - independent of hardware platform, operating system, programming language, local representation, etc.
- Standard sets of rules for encoding instances of logical data structures that are specified in abstract notation
  - for the purpose of transmission

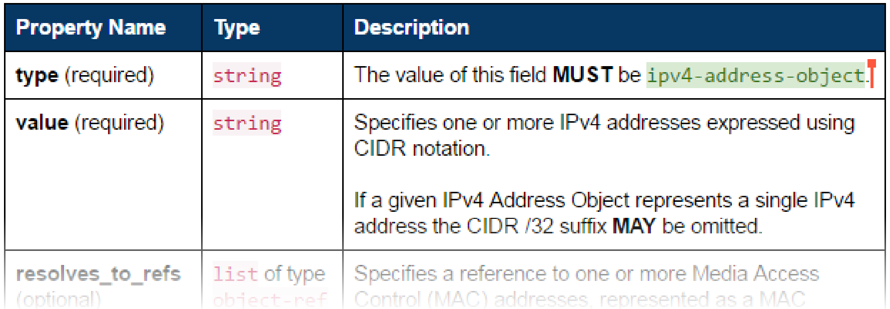
* Description and Principles by Allesandro Trigila
http://www.ieee802.org/802_tutorials/2010-11/
Describes ASN.1, but applies to any abstract syntax language

Principles and Benefits* of abstract syntax

- Separation of concerns
  - The description of the logical structure of a message is kept completely separate from the details of the encoding
- Message descriptions are machine-processable
  - This enables the creation and use of software development tools and testing tools that can read and understand the formal definitions
- Encodings are standardized
  - The problem of specifying detailed encodings and the problem of encoding/decoding messages and their fields do not need to be addressed again and again
- Extensibility
  - It is possible to extend a message description in controlled ways while ensuring backward- and forward-compatibility between different version implementations

OASIS CTI Data Definitions

- STIX, TAXII, CybOX are specified in Property Tables
  - **Abstract description**
    - Independent of serialization (XML, JSON, binary)
  - Informal definition of data objects
    - Cannot be machine parsed, validated, or translated
  - Interim step toward formal abstract specifications
    - Would enable automated translation to concrete schemas:
      - XSD
      - JSON Schema
      - Proto3
      - …



Property Name	Type	Description
type (required)	`string`	The value of this field **MUST** be `ipv4-address-object`.
value (required)	`string`	Specifies one or more IPv4 addresses expressed using CIDR notation. If a given IPv4 Address Object represents a single IPv4 address the CIDR /32 suffix **MAY** be omitted.
resolves_to_refs (optional)	`list` of type `object-ref`	Specifies a reference to one or more Media Access Control (MAC) addresses, represented as a MAC

OpenC2 Data Definitions

- Current implementation: Python abstract data structures
  - Executable: enables encoding-independent message validation
    - Defines OpenC2 abstract syntax
      - Unambiguous type checking
    - Validates example messages
    - Supports multiple message formats
      - JSON (multiple dialects)
      - XML
      - Binary
  - Corresponds directly to informal property tables and formal abstract syntax

```
class OpenC2Command(Record):
    vals = [
        ('action', Action, ''),
        ('target', Target, ''),
        ('actuator', Actuator, '?'),
        ('modifiers', Modifiers, '?')]

class Action(Enumerated):
    vals = [
        'scan',         #  1
        'locate',       #  2
        'query',        #  3
         ...
        'remediate']    # 35

class Target(Record):
    vals = [
        ('type', TargetTypeValue , ''),
        ('specifiers', cybox.CyboxObject,
                                  '?,{type}')]
```

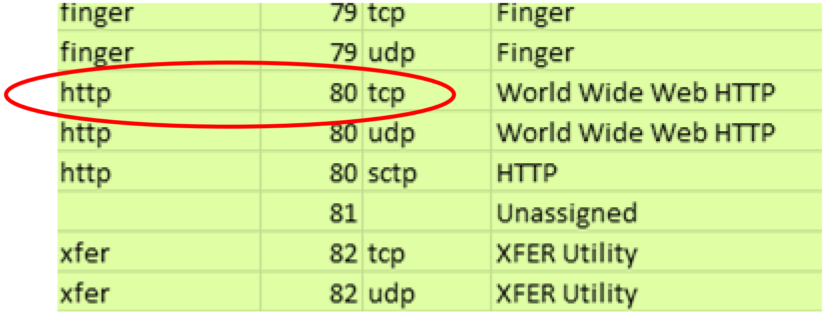
Abstract Syntax - Structures

- JSON Data Model
  - Array – ordered list of items
    - Item has position, no name
  - Object – unordered set of properties
    - Property has name, no position
- Abstract Data Model
  - Record – ordered list of fields
    - Field has both name and position
    - Encoded in JSON as either Array or Object
    - Decoder restores names to Array fields, positions to Object fields
  - Map – unordered set of fields
    - Field has name, no position
    - Encoded in JSON as Object

```
class OpenC2Command(Record):
    vals = [
        ('action', Action, ''),
        ('target', Target, ''),
        ('actuator', Actuator, '?'),
        ('modifiers', Modifiers, '?')]
```

Abstract Syntax – Names

- JSON Data Model
  - Names transmitted as strings
    - Field names / property keys (e.g., “type”, “value”, “Action”)
    - Literals in a vocabulary (e.g., “ipv4-address-object”, “TCP”, “scan”)
- Abstract Data Model
  - Names transmitted as either strings or tags
    - Tags (ElementIDs) assigned to Names in a registry
    - Example - port numbers:

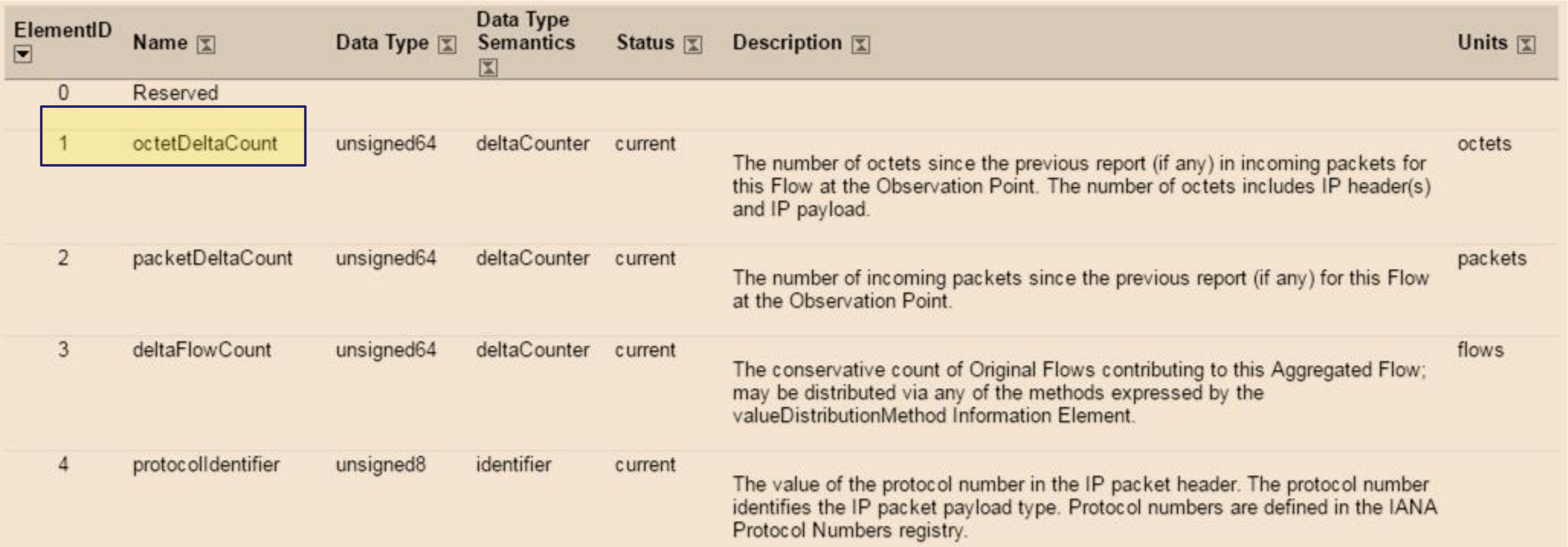


finger	79	tcp	Finger
finger	79	udp	Finger
http	80	tcp	World Wide Web HTTP
http	80	udp	World Wide Web HTTP
http	80	sctp	HTTP
	81		Unassigned
xfer	82	tcp	XFER Utility
xfer	82	udp	XFER Utility

```
class Action(Enumerated):
    vals = [
        'scan',        #  1
        'locate',      #  2
        'query',       #  3
         ...
        'remediate']   # 35
```

Abstract Syntax – Names (cont.)

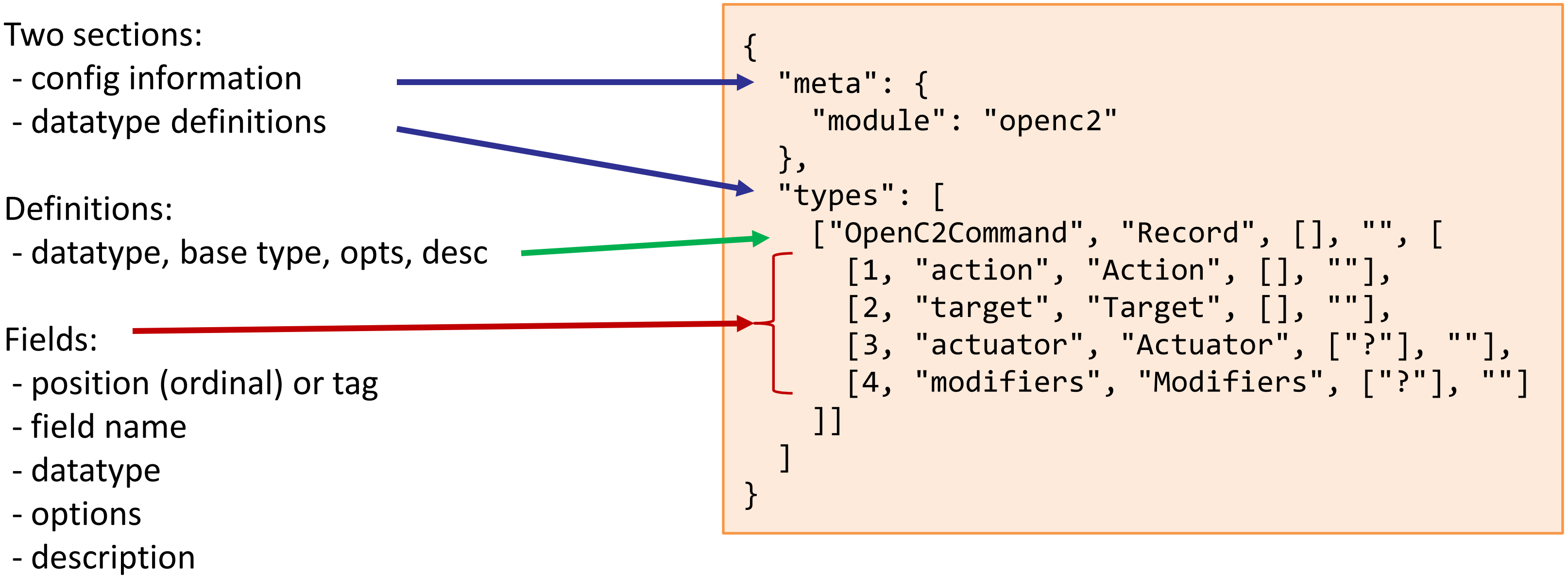
- IP Flow Information Export (IPFIX) element registry
  - Abstract syntax assigns IDs (1) to names (“octetDeltaCount”)
  - Concrete encoding uses one or the other
  - Decoder supplies name corresponding to received ID
  - Namespace used to identify registry



ElementID	Name	Data Type	Data Type Semantics	Status	Description	Units
0	Reserved					
1	octetDeltaCount	unsigned64	deltaCounter	current	The number of octets since the previous report (if any) in incoming packets for this Flow at the Observation Point. The number of octets includes IP header(s) and IP payload.	octets
2	packetDeltaCount	unsigned64	deltaCounter	current	The number of incoming packets since the previous report (if any) for this Flow at the Observation Point.	packets
3	deltaFlowCount	unsigned64	deltaCounter	current	The conservative count of Original Flows contributing to this Aggregated Flow; may be distributed via any of the methods expressed by the valueDistributionMethod Information Element.	flows
4	protocolIdentifier	unsigned8	identifier	current	The value of the protocol number in the IP packet header. The protocol number identifies the IP packet payload type. Protocol numbers are defined in the IANA Protocol Numbers registry.	

JSON Abstract Encoding Notation (JAEN)

- JSON document that defines an abstract schema
  - Import directly by applications, or
  - Translate to concrete schemas used by applications



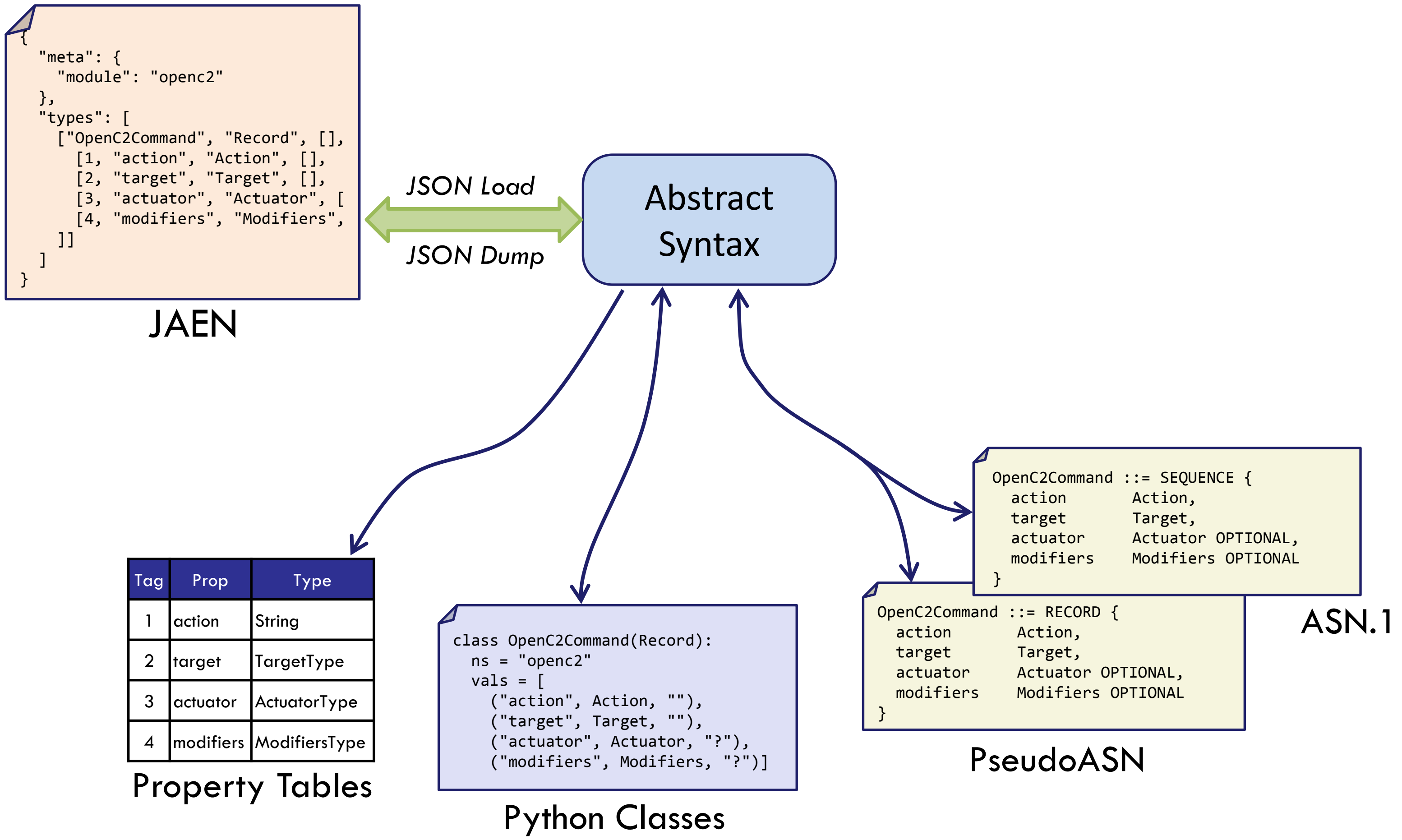
Two sections:
- config information
- datatype definitions

Definitions:
- datatype, base type, opts, desc

Fields:
- position (ordinal) or tag
- field name
- datatype
- options
- description

```
{
  "meta": {
    "module": "openc2"
  },
  "types": [
    ["OpenC2Command", "Record", [], "", [
      [1, "action", "Action", [], ""],
      [2, "target", "Target", [], ""],
      [3, "actuator", "Actuator", ["?"], ""],
      [4, "modifiers", "Modifiers", ["?"], ""]
    ]]
  ]
}
```

Abstract Schema Representations



```
{
  "meta": {
    "module": "openc2"
  },
  "types": [
    ["OpenC2Command", "Record", [],
      [1, "action", "Action", [],
      [2, "target", "Target", [],
      [3, "actuator", "Actuator", [
      [4, "modifiers", "Modifiers",
    ]]
  ]
}
```

JAEN

Tag	Prop	Type
1	action	String
2	target	TargetType
3	actuator	ActuatorType
4	modifiers	ModifiersType

Property Tables

```
class OpenC2Command(Record):
  ns = "openc2"
  vals = [
    ("action", Action, ""),
    ("target", Target, ""),
    ("actuator", Actuator, "?"),
    ("modifiers", Modifiers, "?")]
```

Python Classes

```
OpenC2Command ::= SEQUENCE {
  action        Action,
  target        Target,
  actuator      Actuator OPTIONAL,
  modifiers     Modifiers OPTIONAL
}
```

ASN.1

```
OpenC2Command ::= RECORD {
  action        Action,
  target        Target,
  actuator      Actuator OPTIONAL,
  modifiers     Modifiers OPTIONAL
}
```

PseudoASN

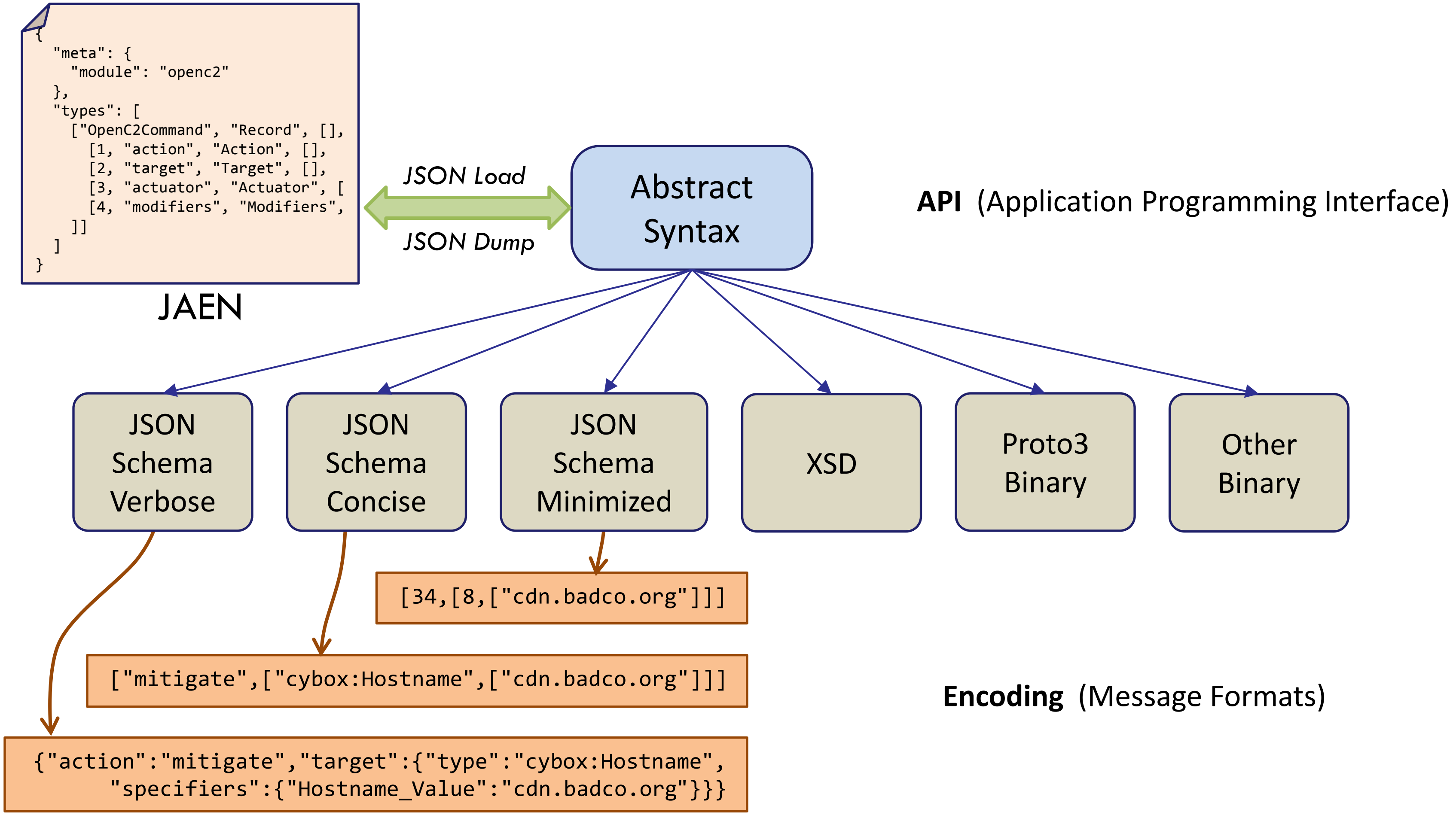
Pseudo-ASN (PASN)

- Patterned after ASN.1, but modified for ease of use
  - ASN.1 has no first-class map (key:value pair) type
    - SEQUENCE / SEQUENCE OF and SET / SET OF are the only compound ASN.1 types
    - Table Constraint syntax is general but cumbersome
    - PASN defines MAP to represent Identifier:Typereference pairs
  - ASN.1 restricts case for Identifier and Typereference
    - PASN allows both upper and lower case first character
  - ASN.1 SEQUENCE does not support encoding modes
    - JSON Encoding Rules (to be defined) might add encoding modes
    - PASN defines RECORD to be encoded as either JSON object or array
  - PASN requires explicit tags for names
    - RECORD field tags are optional, must be ordinal if present
  - PASN does not allow anonymous type definitions
    - Fields contain references to named types
    - Supports direct translation to JAEN without compiling

JSON Encoding Modes

- Verbose
  - RECORD encoded as Object
  - Highest bandwidth
  - Arguably most human-readable (explicit field names)
- Concise
  - RECORD encoded as Array
  - Reduced bandwidth
  - Arguably more readable (no field name clutter)
- Minimized
  - RECORD encoded as Array, Names encoded as Tags
  - Most bandwidth efficient, least readable
  - Use directly for transmission, or as visualization of binary encoding

Concrete Schema Generation



```
{
  "meta": {
    "module": "openc2"
  },
  "types": [
    ["OpenC2Command", "Record", [],
      [1, "action", "Action", [],
      [2, "target", "Target", [],
      [3, "actuator", "Actuator", [
      [4, "modifiers", "Modifiers",
    ]]
  ]
}
```

JAEN

JSON Load

JSON Dump

Abstract Syntax

API (Application Programming Interface)

JSON Schema Verbose

JSON Schema Concise

JSON Schema Minimized

XSD

Proto3 Binary

Other Binary

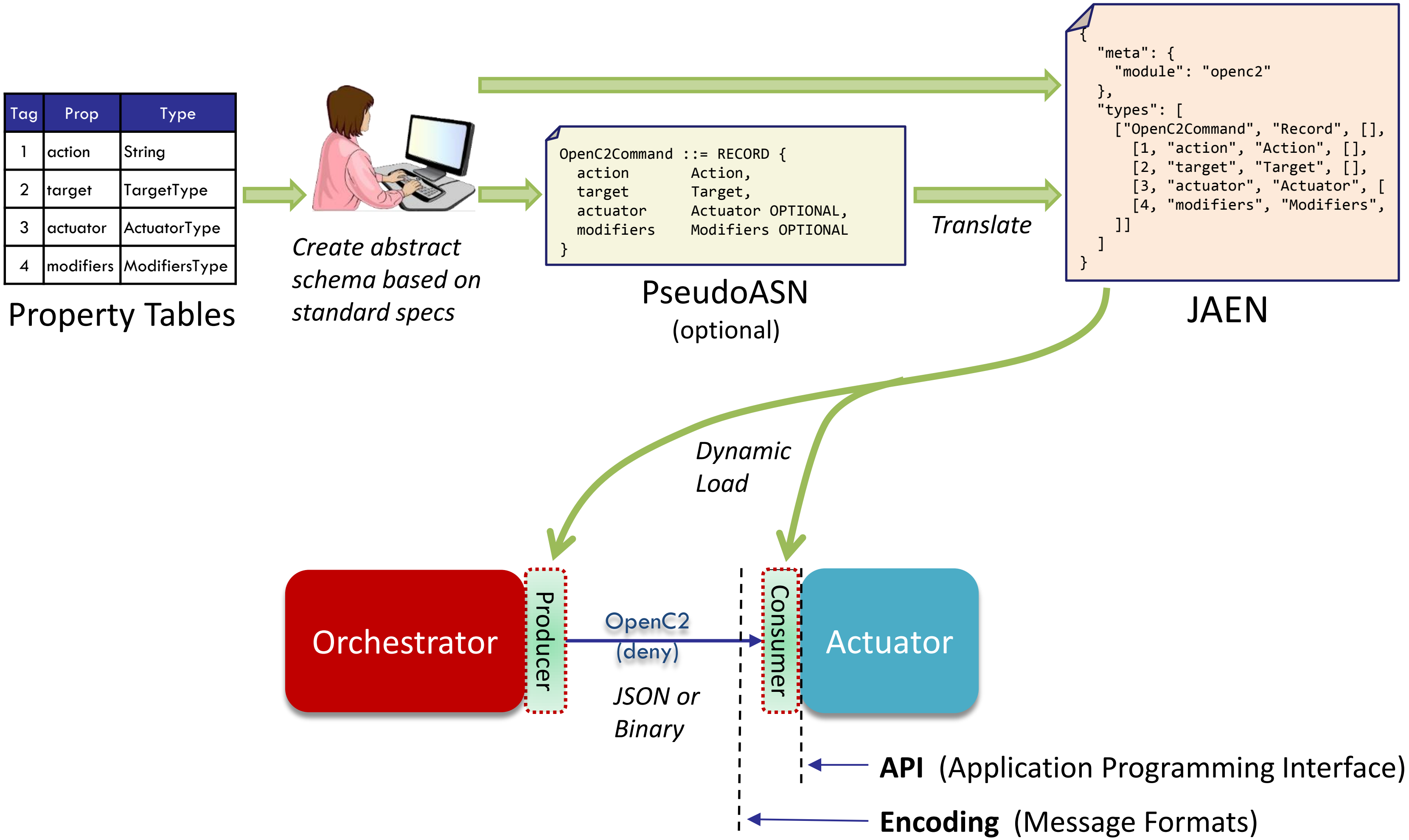
```
[34,[8,["cdn.badco.org"]]]
```

```
["mitigate",["cybox:Hostname",["cdn.badco.org"]]]
```

```
{"action":"mitigate","target":{"type":"cybox:Hostname",
      "specifiers":{"Hostname_Value":"cdn.badco.org"}}}
```

Encoding (Message Formats)

Application Development



Tag	Prop	Type
1	action	String
2	target	TargetType
3	actuator	ActuatorType
4	modifiers	ModifiersType

Property Tables

Create abstract schema based on standard specs

```
OpenC2Command ::= RECORD {
  action        Action,
  target        Target,
  actuator      Actuator OPTIONAL,
  modifiers     Modifiers OPTIONAL
}
```

PseudoASN (optional)

Translate

```
{
  "meta": {
    "module": "openc2"
  },
  "types": [
    ["OpenC2Command", "Record", [],
      [1, "action", "Action", [],
      [2, "target", "Target", [],
      [3, "actuator", "Actuator", [
      [4, "modifiers", "Modifiers",
    ]]
  ]
}
```

JAEN

Dynamic Load

JSON or Binary

API (Application Programming Interface)

Encoding (Message Formats)

Message Structure

- API supports structure (nested objects) and template (object with path keys) formats
  - Identical information, lossless bidirectional conversion
  - Template format may be easier for applications to work with

```
{ "ACTION": "DENY",
  "TARGET": {
    "type": "cybox:Network_Connection",
    "specifiers": {
      "Layer3Protocol": "IPv4",
      "Layer4Protocol": "TCP",
      "SourceSocketAddress": {
        "IP_Address": {
          "Address_Value": "any"}},
      "DestinationSocketAddress": {
        "IP_Address": {
          "Address_Value": "10.10.10.2"}}}},
  "ACTUATOR": {
    "type": "network-firewall",
    "specifiers": {
      "asset_id": "30"}},
  "MODIFIERS": {
    "context_ref": 91}
}
```

```
{ "ACTION": "DENY",
  "TARGET.type": "cybox:Network_Connection",
  "TARGET.specifiers.Layer3Protocol": "IPv4",
  "TARGET.specifiers.Layer4Protocol": "TCP",
  "TARGET.specifiers.SourceSocketAddress.IP_Address.Address_Value": "any",
  "TARGET.specifiers.DestinationSocketAddress.IP_Address.Address_Value":
"10.10.10.2",
  "ACTUATOR.type": "network-firewall",
  "ACTUATOR.specifiers.asset_id": "30",
  "MODIFIERS.context_ref": 91
}
```

Lessons Learned

- Distributed assignment (namespaces):
  - No generally-accepted namespace approach for JSON
    - Forced to roll our own to use both CybOX 2 and CybOX 3
    - Need standardized namespace approach
- Balance between nesting and referencing:
  - STIX 1 allowed structures with unlimited nesting levels
  - STIX 2 (and CybOX 3) forbid nesting entirely
    - Result: IP Address object uses a reference (pointer) to a MAC Address object, Excessive message overhead for containers, References complicate message definition and validation
    - Everything in moderation – allow 1-2 nesting levels, but not unlimited

Lessons Learned (cont.)

- Think Abstract!
  - Designers need names to understand data
    - Result: Protocols defined at transport level send names, wasting bandwidth
    - Design at abstract level (write message APIs in “source code”)
      - Communicate using efficient concrete data and schemas (“machine code”)
      - Avoid mistakes like using objects to emulate arrays
    - Message templates can ease application integration
  - Tools can use abstract schema
    - Menu-based or template-based message composers
    - “Wireshark” display module
  - Applications should provide mechanism, not policy
    - Producer selects message encoding at runtime
    - Consumer config’ed to accept one or many message encodings
    - Strict (lint) or permissive (case-insensitive) receive modes

Next Steps

- Initial Python codec release
  - Current code is incomplete, alpha level proof-of-concept
    - Need JAEN-based decoder
    - Need encoder methods
    - Need CybOX 3 JAEN definitions
    - Need concrete schema generator
    - Need test suite
    - Need documentation
    - Need …
- Socialize abstract design approach with CTI