Information Modeling
with JADN

David Kemp

OpenC2 TC Face-to-Face
1 May 2019

Purpose

- **JADN** (JSON Abstract Data Notation) is a language for defining **Information Models**.
- What is Information?
- What is an Information Model?

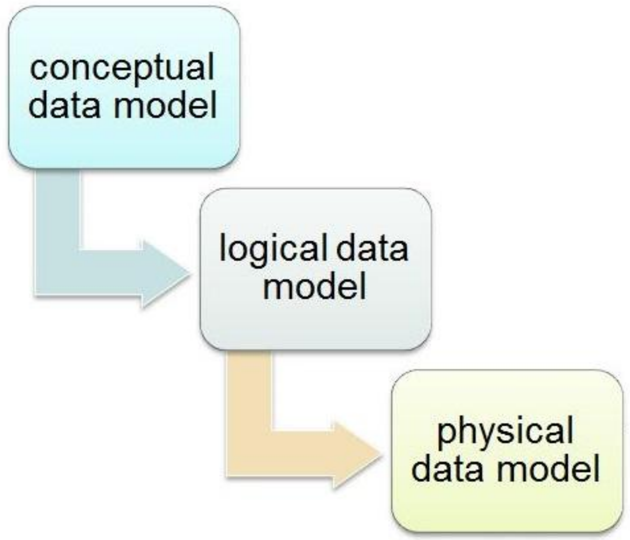
Information

- *What is conveyed or represented by a particular arrangement or sequence of things.*
  "genetically transmitted information"
- *Information is the resolution of uncertainty; it is that which answers the question of "what an entity is" and is thus that which specifies the nature of that entity.*
    --- Wikipedia

Information Model

```
The main purpose of an IM is to model managed objects
at a conceptual level, independent of any specific
implementations or protocols used to transport the
data.  The degree of specificity (or detail) of the
abstractions defined in the IM depends on the modeling
needs of its designers.

 --- RFC 3444: "On the Difference between Information
                Models and Data Models"
```



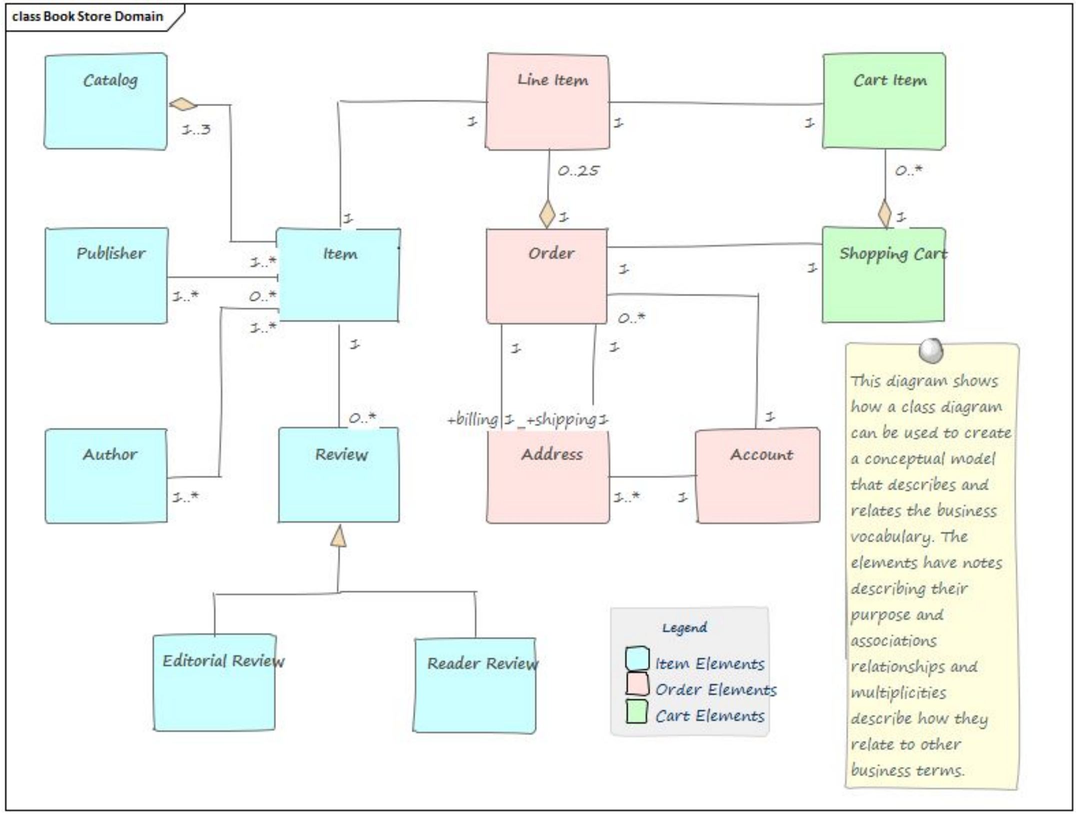
Information Model sometimes refers to conceptual models represented graphically with entity relationship diagrams.

Here, ***Information Model*** is a middle level of abstraction. Structures, fields, value constraints and semantics are fully defined, but the definitions are independent of data format.

A ***Data Model*** includes protocol- and implementation-specific details, including data format.

Conceptual Model

(Highest level of abstraction, not what we mean by IM)



This diagram shows how a class diagram can be used to create a conceptual model that describes and relates the business vocabulary. The elements have notes describing their purpose and associations relationships and multiplicities describe how they relate to other business terms.

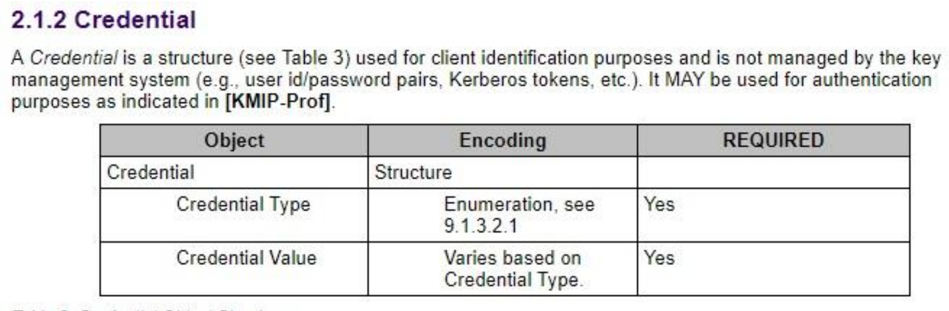
https://sparxsystems.com/enterprise_architect_user_guide/14.0/guidebooks/ea_conceptual_information_model.html

Information Model

(Middle level of abstraction)

- KMIP defines multiple data models (TTLV, XML, JSON)
- STIX 1 has one DM (XML)
- STIX 2 has one DM (JSON) but claims to "support additional serializations"

Key Management Interoperability Protocol (KMIP):

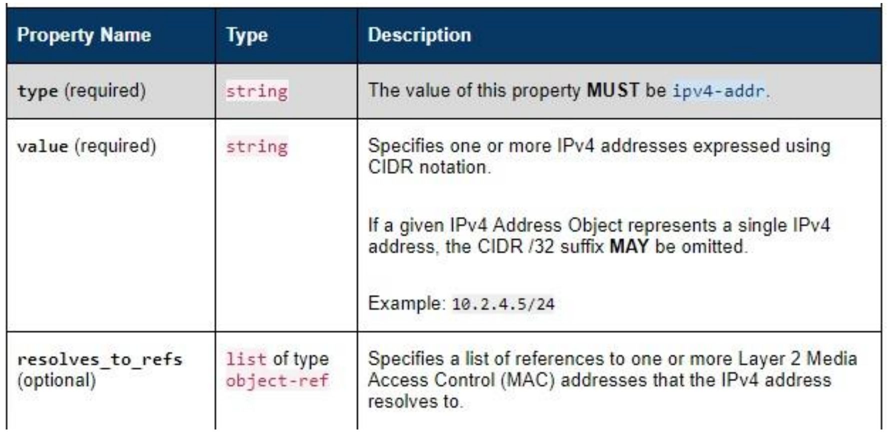


2.1.2 Credential

A *Credential* is a structure (see Table 3) used for client identification purposes and is not managed by the key management system (e.g., user id/password pairs, Kerberos tokens, etc.). It MAY be used for authentication purposes as indicated in **[KMIP-Prof]**.

Object	Encoding	REQUIRED
Credential	Structure	
Credential Type	Enumeration, see 9.1.3.2.1	Yes
Credential Value	Varies based on Credential Type.	Yes

Structured Threat Information Expression (STIX):



Property Name	Type	Description
`type` (required)	`string`	The value of this property **MUST** be `ipv4-addr`.
`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. Example: `10.2.4.5/24`
`resolves_to_refs` (optional)	`list` of type `object-ref`	Specifies a list of references to one or more Layer 2 Media Access Control (MAC) addresses that the IPv4 address resolves to.

Why use an IM Language?

- Define the "nature" of whatever applications need to convey (e.g., an IPv4 address, a capability structure)
  - Answer any application question: *"What is the TTL of this packet?"*
- Serialize it using any data format, based on communication priorities:
  - Verbose JSON, XML
  - Compact JSON, CBOR, IP
- Separate the intellectual investment in defining application functionality from serialization, which can be commoditized

Information Types

JADN defines three Information types (where serialization rules specify Data types)

1. **Enumerated** - A list of ID-Name pairs that define membership in a category. Data = either ID or Name.
2. **Record** - A row of values whose type is specified by position / column. Data = either Array or Map.
3. **Map** - a map from key to value where keys are Enumerated. Data = map/dict/object where keys are either ID or Name.

Enumerated

POSIX: #include <fcntl.h>

Defines the "nature" of open flags - what is and is not an open flag, and what each valid flag means.

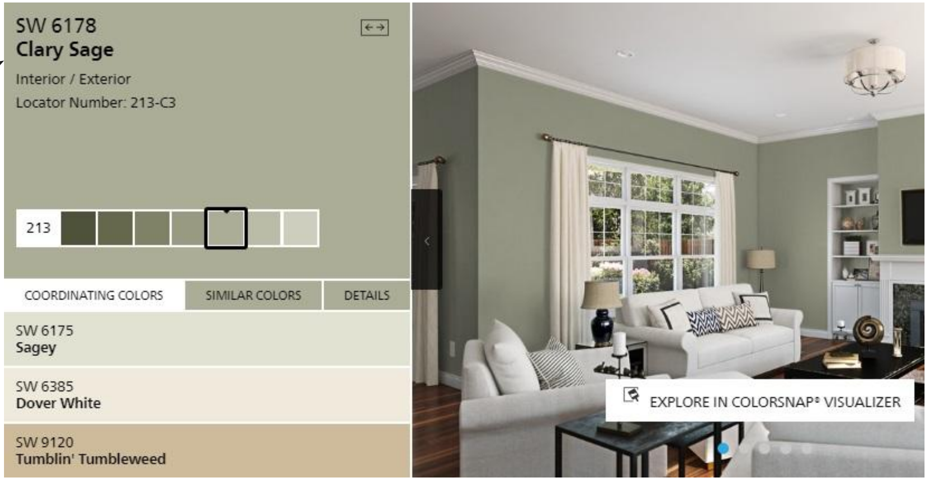
0x0000 means:

- *read only*
- *lecture seulement*
- *μόνο για ανάγνωση*

O_SHUFFLE and 0x0800 are not open flags.

```
/*
 * File status flags: these are used by open(2), fcntl(2).
 */
/* open-only flags */
#define  O_RDONLY   0x0000  /* open for reading only */
#define  O_WRONLY   0x0001  /* open for writing only */
#define  O_RDWR     0x0002  /* open for reading and writing */
#define  O_ACCMODE  0x0003  /* mask for above modes */
```

Enumerated



SW 6178
Clary Sage
Interior / Exterior
Locator Number: 213-C3

213

COORDINATING COLORS | SIMILAR COLORS | DETAILS

SW 6175
Sagey

SW 6385
Dover White

SW 9120
Tumblin' Tumbleweed

EXPLORE IN COLORSNAP® VISUALIZER

The Sherwin-Williams catalog defines the "nature" of their paint colors - what is and is not a valid color.

"Clary Sage" is a color.
"Claret" is not a color.
"Fourscore and seven years ago" is not a color.

Information = specific thing that is on the list.

Data = either ID (16 bits) or string (32 characters / 256 bits?).

Non-enumerated data = any string.

Information vs. Data

RFC 791 uses a data format that carries just the **Information** needed by network applications.

How do we know?   One hint:

> The RFC can be translated into other languages, and software developers who follow the specification will create **interoperable** implementations. The data in an IPv4 packet **does not depend** on the language in which the specification is written.

Another hint:

> The IPv4 address space has 4,294,967,296 (2**32) unique addresses. The IP data format takes the minimum number of bits (32) required to designate any possible address*. Taking up more bits of data would not increase the number of network addresses.

* Without considering non-uniform probability distributions

IPv4 - English

https://tools.ietf.org/html/rfc791#section-3.1

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

IPv4 - German

Are these the same protocol? Yes.

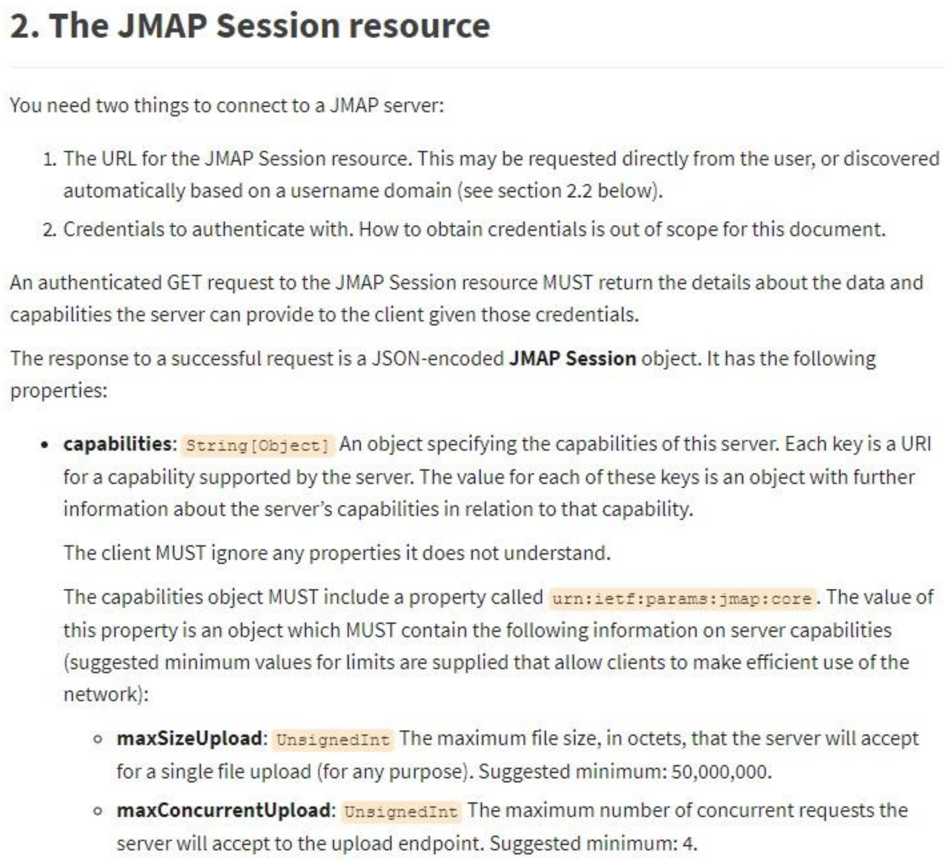
- IPv4 is language-agnostic because the IP **data** format is boiled down to contain only **information** with no additional overhead.
- JSON data formats can be language-specific (non-interoperable) or language-agnostic (interoperable).

IPv4 Implementations built by programmers of any nationality following a spec written in any language will interoperate.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
```

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Ausführ|  IHL  |Art der Dienstl|          Gesamtlänge          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identifikation        |Flagg|   Fragmentversatz       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Zeit zu leben |   Protokoll   |         Header Prüfsumme      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Quelladresse                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Zieladresse                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Optionen                |    Auffüllen  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

JMAP - English



2. The JMAP Session resource

You need two things to connect to a JMAP server:

1. The URL for the JMAP Session resource. This may be requested directly from the user, or discovered automatically based on a username domain (see section 2.2 below).
2. Credentials to authenticate with. How to obtain credentials is out of scope for this document.

An authenticated GET request to the JMAP Session resource MUST return the details about the data and capabilities the server can provide to the client given those credentials.

The response to a successful request is a JSON-encoded **JMAP Session** object. It has the following properties:

- **capabilities**: `String[Object]` An object specifying the capabilities of this server. Each key is a URI for a capability supported by the server. The value for each of these keys is an object with further information about the server's capabilities in relation to that capability.

  The client MUST ignore any properties it does not understand.

  The capabilities object MUST include a property called `urn:ietf:params:jmap:core`. The value of this property is an object which MUST contain the following information on server capabilities (suggested minimum values for limits are supplied that allow clients to make efficient use of the network):

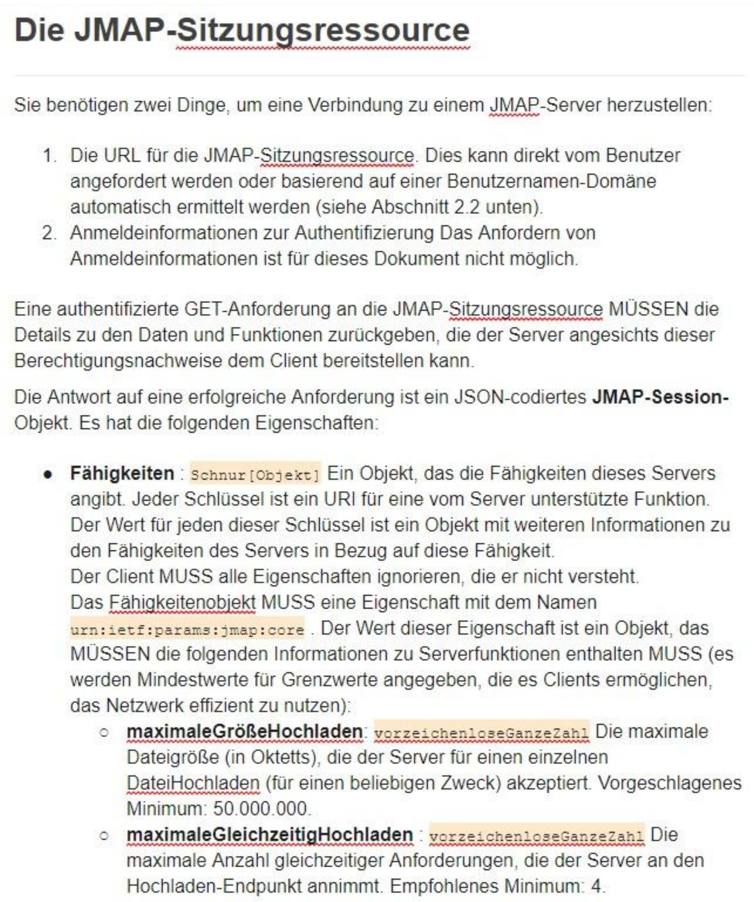
  - **maxSizeUpload**: `UnsignedInt` The maximum file size, in octets, that the server will accept for a single file upload (for any purpose). Suggested minimum: 50,000,000.
  - **maxConcurrentUpload**: `UnsignedInt` The maximum number of concurrent requests the server will accept to the upload endpoint. Suggested minimum: 4.

JMAP - German

Are these the same protocol? No.

- Because JMAP is written as a data specification, not an information specification, it cannot be translated without destroying interoperability.

JMAP implementations built by programmers of any nationality must follow an English-language spec in order to interoperate.



Die JMAP-Sitzungsressource

Sie benötigen zwei Dinge, um eine Verbindung zu einem JMAP-Server herzustellen:

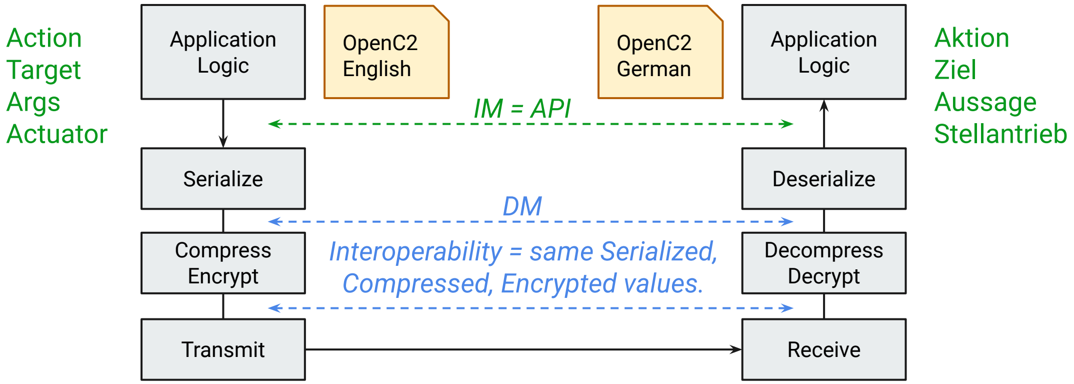
1. Die URL für die JMAP-Sitzungsressource. Dies kann direkt vom Benutzer angefordert werden oder basierend auf einer Benutzernamen-Domäne automatisch ermittelt werden (siehe Abschnitt 2.2 unten).
2. Anmeldeinformationen zur Authentifizierung Das Anfordern von Anmeldeinformationen ist für dieses Dokument nicht möglich.

Eine authentifizierte GET-Anforderung an die JMAP-Sitzungsressource MÜSSEN die Details zu den Daten und Funktionen zurückgeben, die der Server angesichts dieser Berechtigungsnachweise dem Client bereitstellen kann.

Die Antwort auf eine erfolgreiche Anforderung ist ein JSON-codiertes **JMAP-Session-**Objekt. Es hat die folgenden Eigenschaften:

- **Fähigkeiten** : `Schnur[Objekt]` Ein Objekt, das die Fähigkeiten dieses Servers angibt. Jeder Schlüssel ist ein URI für eine vom Server unterstützte Funktion. Der Wert für jeden dieser Schlüssel ist ein Objekt mit weiteren Informationen zu den Fähigkeiten des Servers in Bezug auf diese Fähigkeit.
  Der Client MUSS alle Eigenschaften ignorieren, die er nicht versteht.
  Das Fähigkeitenobjekt MUSS eine Eigenschaft mit dem Namen `urn:ietf:params:jmap:core` . Der Wert dieser Eigenschaft ist ein Objekt, das MÜSSEN die folgenden Informationen zu Serverfunktionen enthalten MUSS (es werden Mindestwerte für Grenzwerte angegeben, die es Clients ermöglichen, das Netzwerk effizient zu nutzen):
  - **maximaleGrößeHochladen**: `vorzeichenloseGanzeZahl` Die maximale Dateigröße (in Oktetts), die der Server für einen einzelnen DateiHochladen (für einen beliebigen Zweck) akzeptiert. Vorgeschlagenes Minimum: 50.000.000.
  - **maximaleGleichzeitigHochladen** : `vorzeichenloseGanzeZahl` Die maximale Anzahl gleichzeitiger Anforderungen, die der Server an den Hochladen-Endpunkt annimmt. Empfohlenes Minimum: 4.

Why use an IM Language? (cont.)



- Applications can use API values in whatever format is preferred by developers
- Different implementations of same spec can use different API values without affecting interoperability, if an efficient serialization is selected
- Developers don't care what encrypted data looks like; by the same principle they shouldn't care what serialized data looks like as long as it is well-defined

Why use an IM Language? (cont.)

- Using an efficient serialization has two benefits:
  - Smaller messages, faster processing
    - Needed for both constrained nodes/environments and high-volume applications
  - Avoids protocol parochialism
- Verbose and efficient serializations coexist in a single protocol spec, interoperate through either native IM serialization or bridging.

Why use an IM Language? (cont.)

- Check property tables for consistency and correctness
- Fill in the conformance details for property tables
    - STIX mentions supporting multiple serializations but:
        - There is no method for validating that a JSON message conforms to the property tables
        - There is no hint for what a conforming CBOR or XML or IP format message would look like

Why use an IM Language? (cont.)

- Parser and validator can be implemented as layers in the communication stack (analogous to encryption and compression)
- Application-agnostic protocol bridge can translate from one format to another
- Application-agnostic firewall can filter and sanitize data flows
  - get applicable schema
  - validate all messages
  - minimize leakage by (in effect) processing all messages in the most efficient (smallest) serialization format

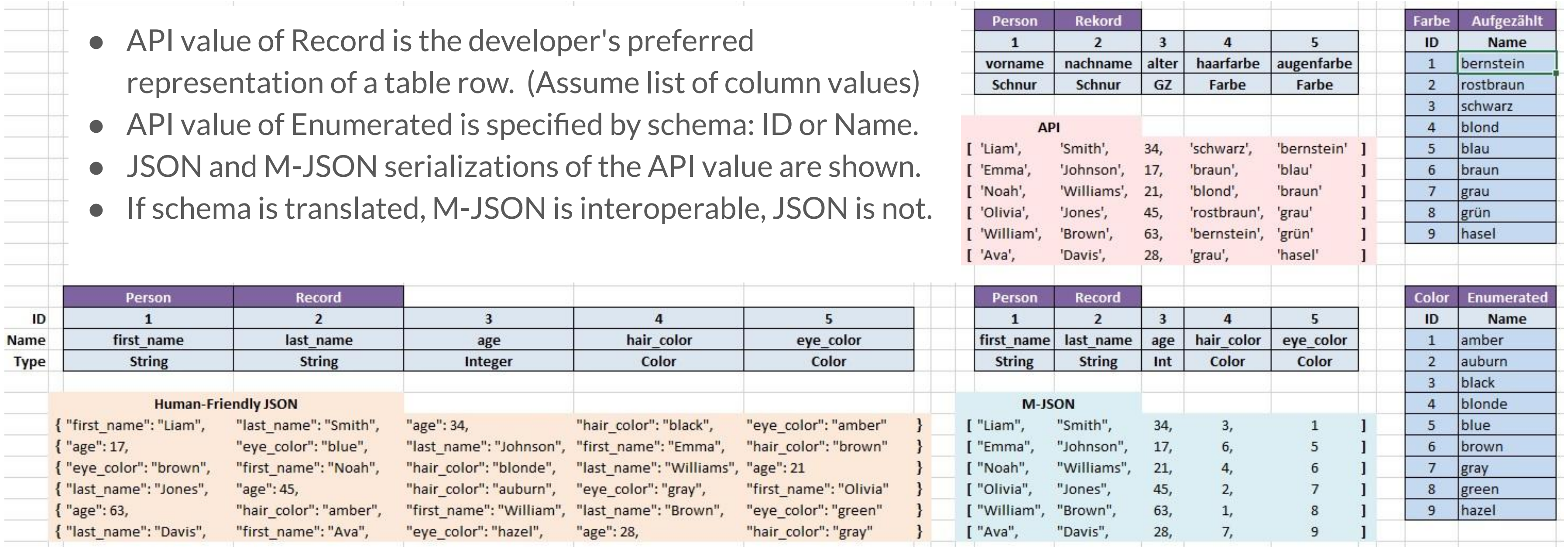
Information Types serialized in JSON

JADN types serialized directly as JSON types: **Boolean, Number, Null, String, Array**

Binary: JSON string containing base64url encoding of the Binary value, or a specialized encoding
Integer: JSON number, restricted to have no fractional part
Enumerated: either JSON number (for ID) or string (for Name)
Map: JSON object where the key is either an ID or a Name
Choice: a Map with exactly one item
ArrayOf: an Array where all values have a specified type
MapOf: a Map where all keys have a specified type and all values have a specified type
Record: either an Array (for compact serializations) or a Map (for verbose serializations)

Table Example: Record, Enumerated types

- API value of Record is the developer's preferred representation of a table row. (Assume list of column values)
- API value of Enumerated is specified by schema: ID or Name.
- JSON and M-JSON serializations of the API value are shown.
- If schema is translated, M-JSON is interoperable, JSON is not.



Person	Rekord			
1	2	3	4	5
vorname	nachname	alter	haarfarbe	augenfarbe
Schnur	Schnur	GZ	Farbe	Farbe

API

['Liam',	'Smith',	34,	'schwarz',	'bernstein'	]
['Emma',	'Johnson',	17,	'braun',	'blau'	]
['Noah',	'Williams',	21,	'blond',	'braun'	]
['Olivia',	'Jones',	45,	'rostbraun',	'grau'	]
['William',	'Brown',	63,	'bernstein',	'grün'	]
['Ava',	'Davis',	28,	'grau',	'hasel'	]

Farbe	Aufgezählt
ID	Name
1	bernstein
2	rostbraun
3	schwarz
4	blond
5	blau
6	braun
7	grau
8	grün
9	hasel

	Person	Record			
ID	1	2	3	4	5
Name	first_name	last_name	age	hair_color	eye_color
Type	String	String	Integer	Color	Color

Human-Friendly JSON

{ "first_name": "Liam",	"last_name": "Smith",	"age": 34,	"hair_color": "black",	"eye_color": "amber"	}
{ "age": 17,	"eye_color": "blue",	"last_name": "Johnson",	"first_name": "Emma",	"hair_color": "brown"	}
{ "eye_color": "brown",	"first_name": "Noah",	"hair_color": "blonde",	"last_name": "Williams",	"age": 21	}
{ "last_name": "Jones",	"age": 45,	"hair_color": "auburn",	"eye_color": "gray",	"first_name": "Olivia"	}
{ "age": 63,	"hair_color": "amber",	"first_name": "William",	"last_name": "Brown",	"eye_color": "green"	}
{ "last_name": "Davis",	"first_name": "Ava",	"eye_color": "hazel",	"age": 28,	"hair_color": "gray"	}

Person	Record			
1	2	3	4	5
first_name	last_name	age	hair_color	eye_color
String	String	Int	Color	Color

M-JSON

["Liam",	"Smith",	34,	3,	1	]
["Emma",	"Johnson",	17,	6,	5	]
["Noah",	"Williams",	21,	4,	6	]
["Olivia",	"Jones",	45,	2,	7	]
["William",	"Brown",	63,	1,	8	]
["Ava",	"Davis",	28,	7,	9	]

Color	Enumerated
ID	Name
1	amber
2	auburn
3	black
4	blonde
5	blue
6	brown
7	gray
8	green
9	hazel

Bob Wehadababyitsaboy

https://www.youtube.com/watch?v=jWPlfWwgFKI

Backup

Additional uses

Downloadable schema enables user interfaces (data explorer with collapsible nodes).

Design by objective:

The proposal for CBOR follows a history of binary formats that is as long as the history of computers themselves. Different formats have had different objectives. **In most cases, the objectives of the format were never stated**, although they can sometimes be implied by the context where the format was first used. Some formats were meant to be universally usable, although history has proven that no binary format meets the needs of all protocols and applications.

CBOR differs from many of these formats due to it **starting with a set of objectives** and attempting to meet just those. This section compares a few of the dozens of formats with CBOR's objectives in order to help the reader decide if they want to use CBOR or a different format for a particular protocol or application.

Design by objective (cont.):

Note that the discussion here is not meant to be a criticism of any format: to the best of our knowledge, no format before CBOR was **meant to cover CBOR's objectives in the priority we have assigned them**:

1. unambiguous encoding of most common data formats from Internet standards
2. code compactness for encoder or decoder
3. no schema description needed
4. reasonably compact serialization
5. applicability to constrained and unconstrained applications
6. good JSON conversion
7. extensibility

JADN Design Objectives

- As with the CBOR data format, JADN follows a long history of schema formats and is meant to address a set of explicitly-stated design objectives.
- JADN is not intended to replace any other schema language. To the contrary, one of its uses is to highlight commonality across schema languages and facilitate translation among them.

JADN Design Objectives

1. Core types represent application-relevant "information", not "data"
2. Single specification unambiguously defines multiple data formats
3. Specification uses named type definitions equivalent to property tables
4. Specification is data that can be serialized
5. Specification has a fixed structure designed for extensibility

1. Types represent Information, not Data

2. Single specification, multiple serializations

A **single** IM specification defines **interoperable** data that meets the **objectives** of multiple data formats:

a. Efficient data, schema-dependent: (IP "bits-in-boxes", Google Protocol Buffers, Facebook/Apache Thrift, SCADA)
b. Reasonably compact data, parse structure without schema: (M-JSON, CBOR, ASN.1 BER)
c. Human-readable text: (JSON, XML)

- *XML in a CBOR text string is technically "CBOR format" but ignores CBOR design objectives.*
- *"All STIX compatible tools must support JSON as a serialization and can, optionally, support additional serializations" ignores interoperability and conformance.*

3. Named types correspond to property tables

Schema uses a flat list of named type definitions

a. Corresponds directly to table-based specifications
b. Corresponds directly to definition-based IDLs (ASN.1, Protobuf, …)
c. JSON Schema is JSON data, but has hierarchical structure
d. YANG is also hierarchical:

```
container system {
  container login {
    leaf message {
      type string;
      description
        "Message given at start of login session";
    }
  }
}
```

A list of named types can be converted to a hierarchy of anonymous types, but the conversion loses information (names) and may result in redundant copies.

4. Schema is structured data

Schema is itself a structured data object that can be serialized

a. Type definitions have a fixed, regular structure
b. Does not require a schema-specific parser
    i. ASN.1, YANG, CDDL, Protobuf, Thrift, etc. are all text languages that must be parsed before use.
    ii. JADN, like JSON Schema, is data that can validate itself.
c. Schema can be embedded in or transmitted with application data
d. Application-agnostic data validation and sanitization

5. Extensible

Future JADN versions can define new capabilities without affecting the type definition structure

New options specify additional details applicable to a type

New fundamental API types, if needed

References

SCADA Communication and Protocols
http://srldc.in/var/NRC/SRLDC%20Fees%20Feb%202013/Trg%20Psti%20System%20Operator%20Trg/PSO%20PSTI%20%205-17%20Sep-2011%20Adisesha/ppts/Protocols/SCADA%20Communications%20and%20Protocols.pdf