

Experiences dealing with hierarchical ENSDF data

Adam Hayes
SG50 Database and Codes
Sept 27 2021

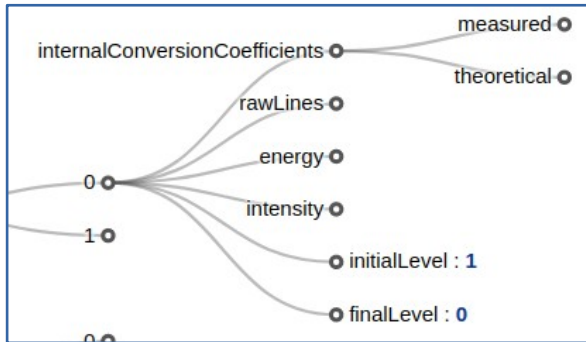
Experiences dealing with hierarchical ENSDF data

- Data representation for hierarchical data
- Comparison of two big players in database management systems
- Efficient validation & testing for our chosen representation
- Side note: graphical visualization

Database choice

ENSDF	My impression of SG-50 / EXFOR
Complex, Hierarchical, Heterogeneous JSON is ideal: <ul style="list-style-type: none">- designed for this type of data- in wide usage- “online” (database) / “offline” (files)- supports binary data	Complex, Hierarchical, Heterogeneous <ul style="list-style-type: none">• JSON has been proposed• Data set content / design have been drafted & discussed
Pure JSON	Viktor proposed a possible hybridization of JSON within relational database
Many data types, including arrays	Many data types, including arrays
Depth of hierarchy? nuclide <ul style="list-style-type: none">→evaluation<ul style="list-style-type: none">→gammas<ul style="list-style-type: none">→gamma 1<ul style="list-style-type: none">→gamma energy datainternal conversion etc. (deep enough that cross-referencing between numerous tables is a pain)	?

Data representation for ENSDF



Hierarchical data can be visualized as a tree with many mixed types

```
"gammaData": {  
  "0": {  
    "rawLines": [...],  
    "energy": [...],  
    "internalConversionCoefficients": {  
      "measured": {...},  
      "theoretical": {...}  
    },  
    "intensity": [...],  
    "initialLevel": 1,  
    "finalLevel": 0  
  },  
  "1": {...}  
}
```

JSON well-suited to tree-like data, easier to read.

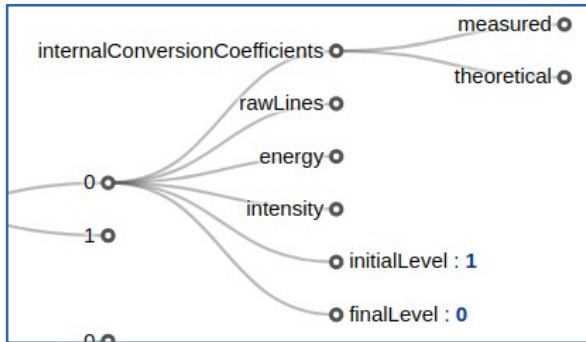
XML works too, but bulkier, especially e.g. for arrays, harder to read

```
<0>  
  <rawLines>  
    ...  
  </rawLines>  
  <energy>  
    ...  
  </energy>  
  <internalConversionCoefficients>  
    <measured>  
      ...  
    </measured>  
  </0>
```

```
...  
  <theoretical>  
    ...  
  </theoretical>  
</internalConversionCoefficients>  
<intensity>  
  ...  
</intensity>  
<initialLevel>1</initialLevel>  
<finalLevel>0</finalLevel>  
</0>  
<1>
```

Both can do the job.

Storage for (JSON) representation?



Hierarchical data can be visualized as a tree with many mixed types

```
"gammaData": {  
  "0": {  
    "rawLines": [...],  
    "energy": {...},  
    "internalConversionCoefficients": {  
      "measured": {...},  
      "theoretical": {...}  
    },  
    "intensity": {...},  
    "initialLevel": 1,  
    "finalLevel": 0  
  },  
  "1": {...}
```

JSON well-suited to tree-like data, easier to read.

Object-oriented databases fully exploit JSON structure

Both can do the job.

Pure relational databases can store JSON as strings

Reasoning for storing JSON as objects in an OODB is somewhat analogous to the reasoning for using data “types” in general:

If all of the data in a relational database (ints, floats, dates,...) are stored as strings, more work is required by the developer, for example, to

- compare values
- select data by date
- etc.

Example →

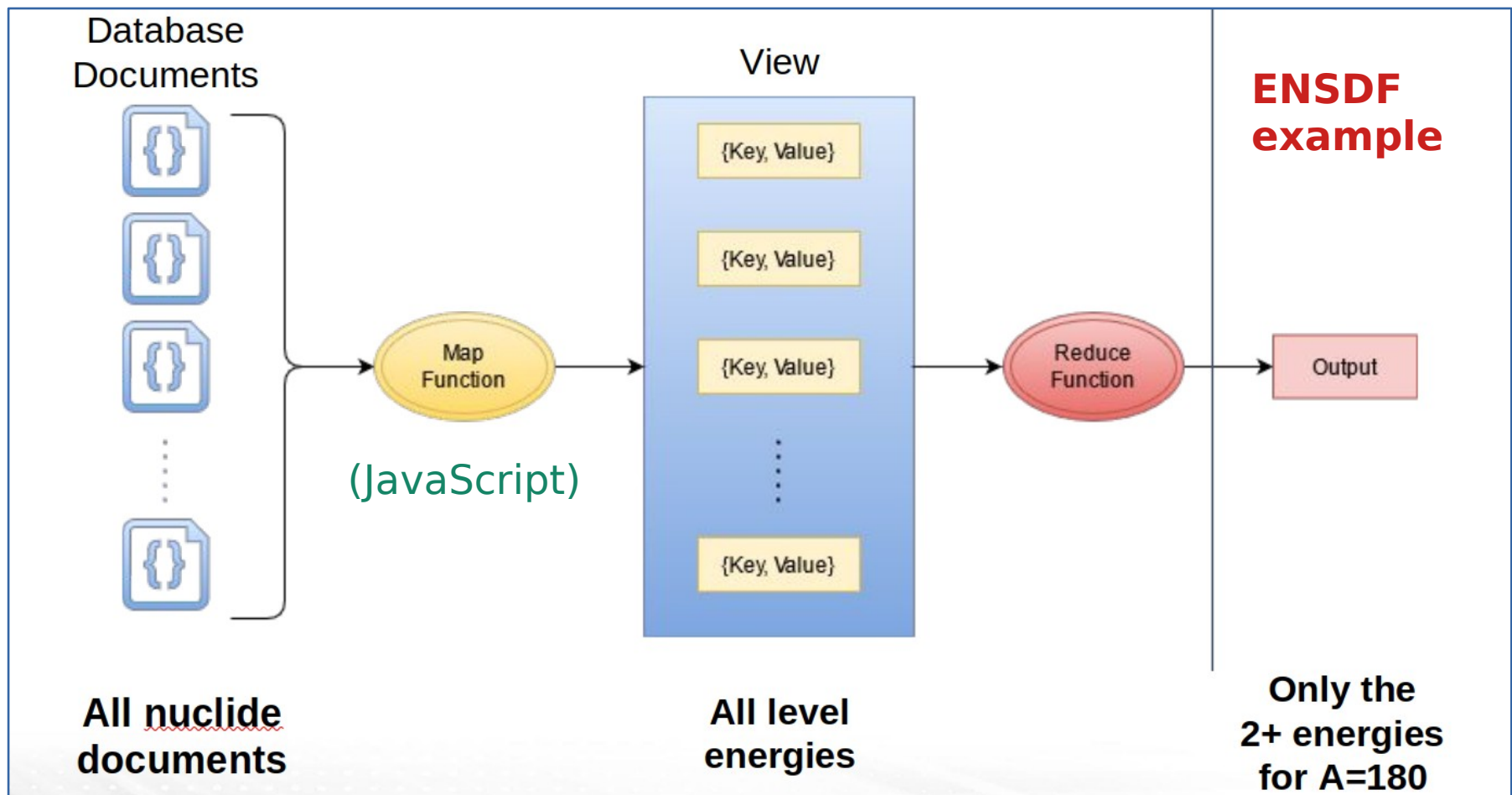
If we store JSON data in files or relational databases, we lose things like...

Views with JSON in CouchDB

Views:

- Filter, search, generate statistics & reports (similar idea to indexing)
- Much more efficient than brute-force search
- Populate once; CouchDB updates **only** for changes, new documents

(CouchDB handles updates *for you*, and it is a built-in feature.)

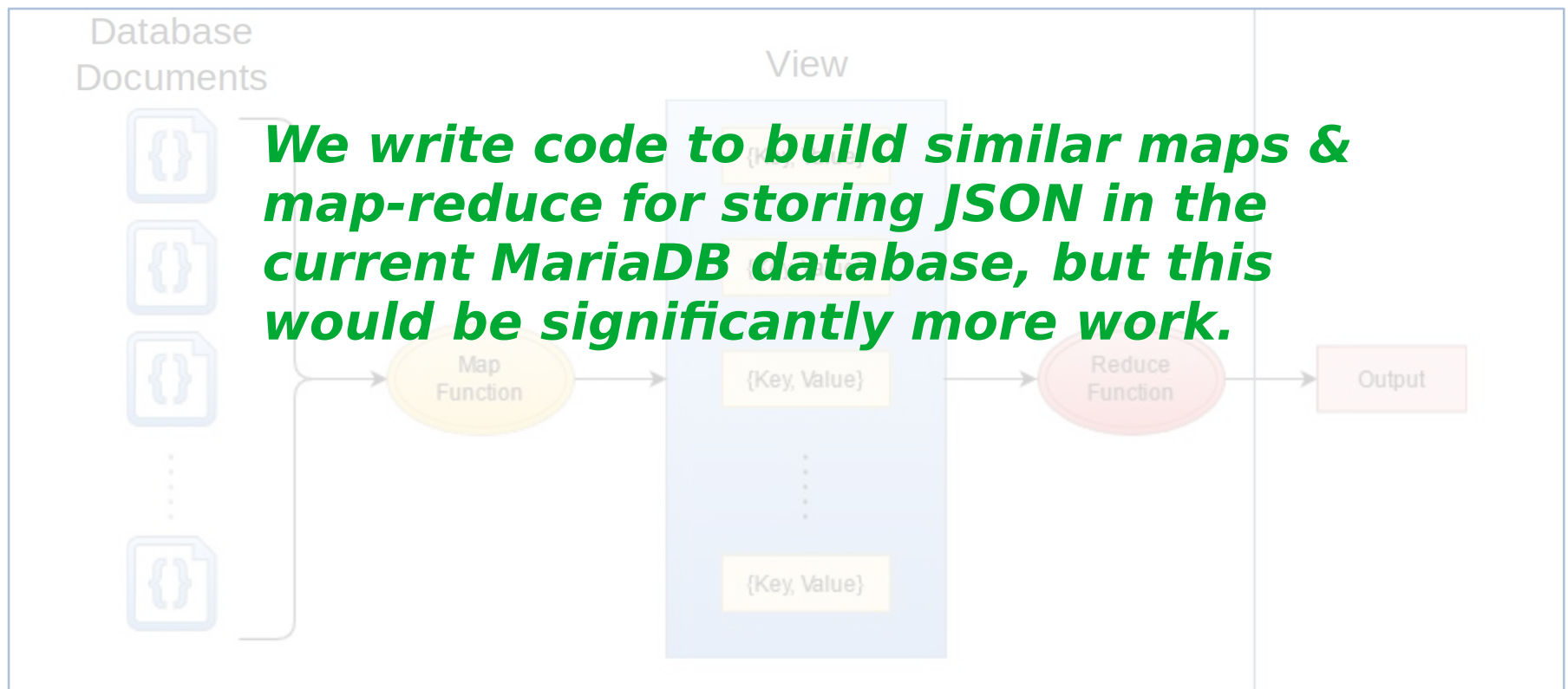


Good example of benefit of OODB for JSON may be: Views with JSON in CouchDB

Views:

- Filter, search, generate statistics & reports (similar idea to indexing)
- Much more efficient than brute-force search
- Populate once; CouchDB updates **only** for changes, new documents

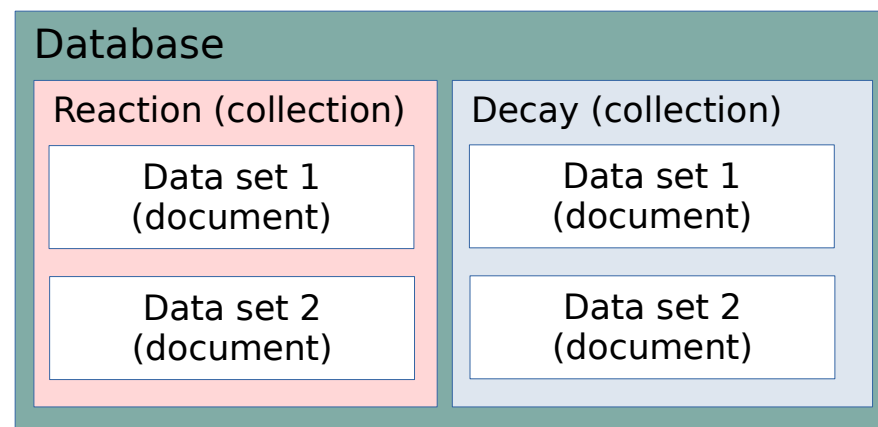
(CouchDB handles updates *for you*, and it is a built-in feature.)



Comparison of two popular OODBMSs

- **MongoDB (2009)**

- eBay, Google, Facebook, PayPal, CERN, bigger market share
- Can be version compatibility issues between API and server
- Uses “collections” of similar-type documents in a database
- Updating a document is allowed
- Massively scalable
- “GridFS” for huge blobs
- APIs for various languages, some better than others
- **Future license terms uncertain**
Current license raises questions—are we providing database “as a service?”
(I don’t think so, but I’m not a lawyer.)

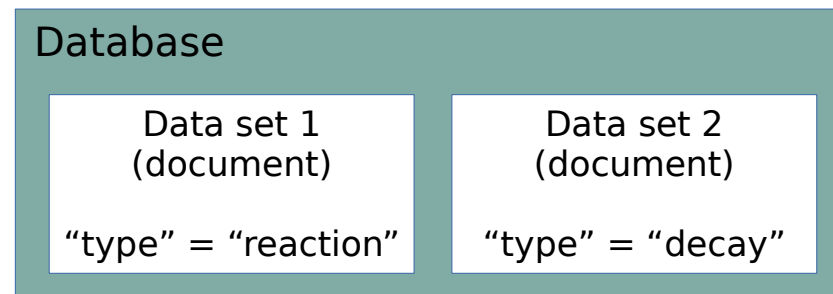


Comparison of two popular OODBMSs

- **CouchDB** (2005)

- Apple, GrubHub, Credit Suisse, Motorola, some Facebook Apps
Smaller market share than Mongo, but large and stable

- No concept of “collections”



- Updating not allowed; modify document and (re)insert

- Massively scalable

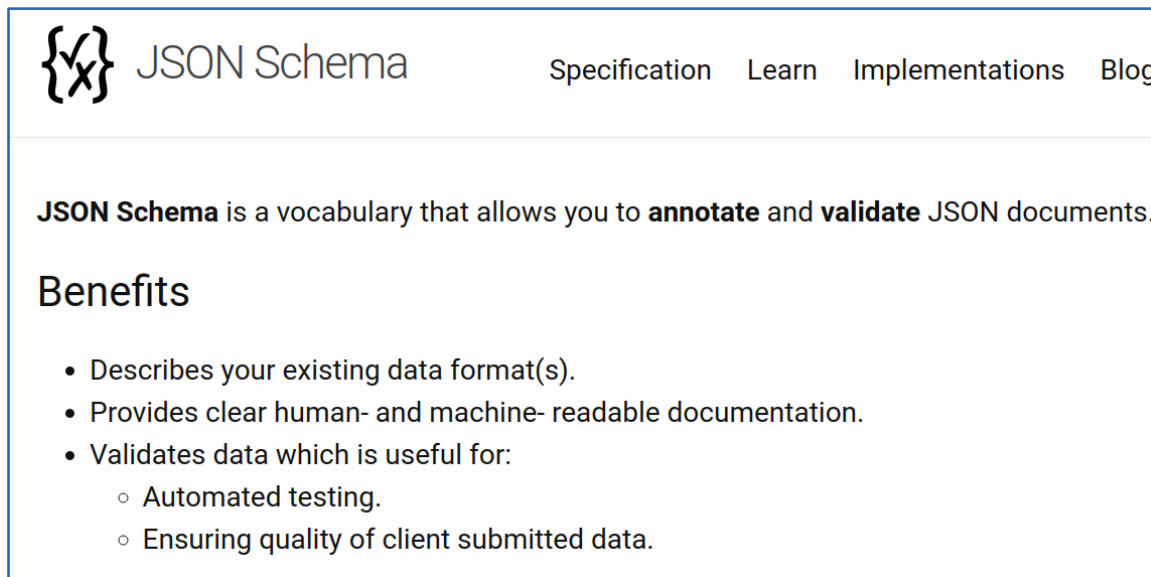
- Can store large (configurable) blobs (or link to outside storage)

- Simple http communication—virtually any language, even Bash.
No reliance on community support for a particular API
<http://db.foo.com:5984/ensdf/137Cs>

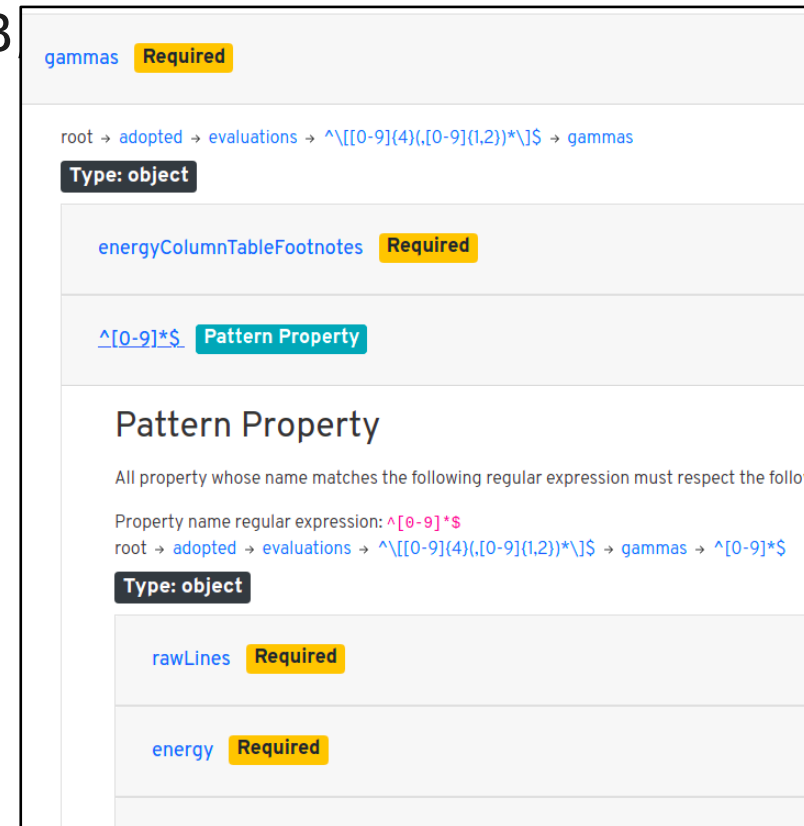
- **Permissive, irrevocable Apache license**

Validation tools with JSON data

- Defines schema & rules for a JSON document
- JSONSchema can even do basic validation of quantitative properties: comparison, if/then, “oneOf” (~case/switch)
- Effective whether you store JSON in an OODB relational DB, or in text files
- Also generates code and documentation →



The screenshot shows the JSON Schema website. At the top left is the logo, a stylized '{X}' inside curly braces. To its right is the text 'JSON Schema'. Further right are navigation links: 'Specification', 'Learn', 'Implementations', and 'Blog'. Below the navigation is a paragraph: 'JSON Schema is a vocabulary that allows you to **annotate** and **validate** JSON documents.' Underneath this is a section titled 'Benefits' with a bulleted list: 'Describes your existing data format(s).', 'Provides clear human- and machine- readable documentation.', and 'Validates data which is useful for:' followed by two sub-bullets: 'Automated testing.' and 'Ensuring quality of client submitted data.'



The screenshot shows a JSON Schema definition for a property named 'gammas'. The path is 'root → adopted → evaluations → ^\[[0-9]{4}\{([0-9]{1,2})*\}\$ → gammas'. The type is 'object'. Below this, there is a 'Pattern Property' section with the regular expression '^([0-9])*\$'. The path for this property is 'root → adopted → evaluations → ^\[[0-9]{4}\{([0-9]{1,2})*\}\$ → gammas → ^([0-9])*\$'. The type is 'object'. Below the pattern property, there are two more properties: 'rawLines' (Required) and 'energy' (Required).

JSONSchema

Official: <https://json-schema.org/>

- Using “JSONSchema” with Python libraries as part of the ENSDF validation

```
{  
  "name": "Mary",  
  "age": 25  
}
```

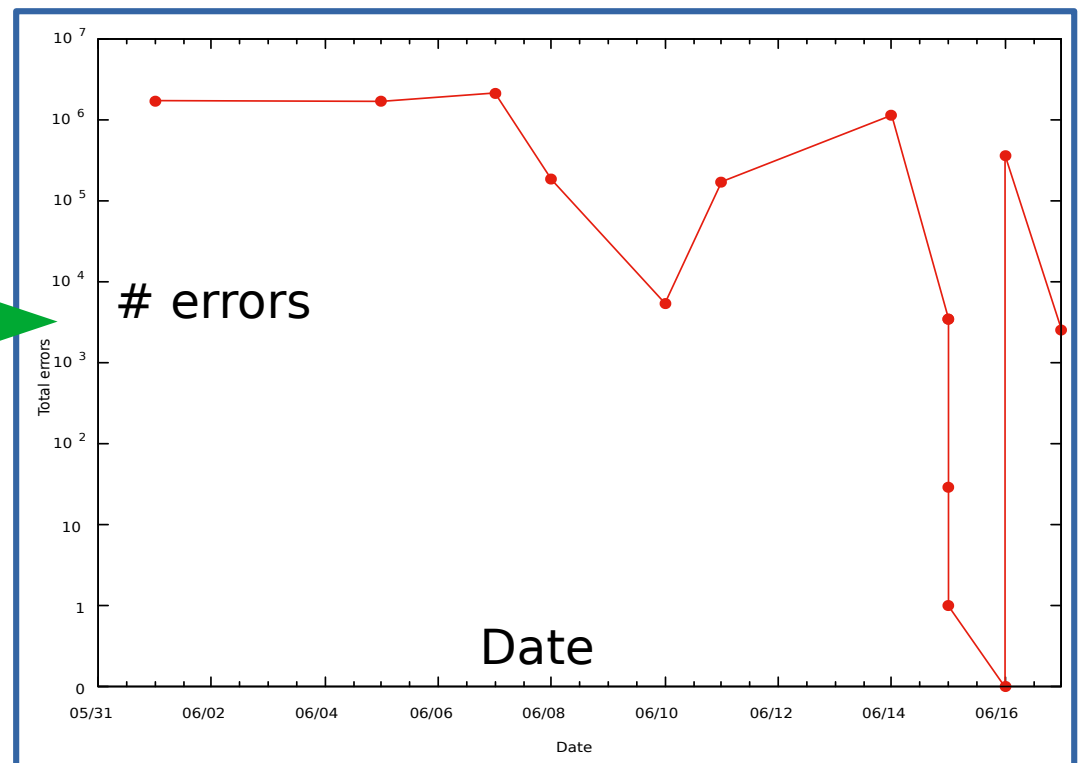
Incoming data

Validation code
e.g. server-side

```
{  
  "properties": {  
    "name": {  
      "type": "string"  
    },  
    "age": {  
      "type": "integer",  
      "minimum": 0  
    }  
  }  
}
```

JSON-Schema
definition

Validates 3400 migrated ENSDF documents in 2 minutes using 15 threads.



Power of JSONSchema

examples from json-schema.org

```
"oneOf": [
  {
    "required": [
      "oneBetaMinusQValue",
      "oneNeutronSeparationEnergy",
      "oneProtonSeparationEnergy"
    ]
  },
  {
    "required": [
      "oneNeutronSeparationEnergy",
      "oneProtonSeparationEnergy",
      "oneBetaPlusQValue"
    ]
  }
]
```

One and only one
of several options

- allOf
- anyOf
- oneOf
- not
- Properties of Schema Composition
 - Subschema Independence
 - Illogical Schemas
 - Factoring Schemas

```
"if": {
  "properties": { "country": { "const": "United States of America" } }
},
"then": {
  "properties": { "postal_code": { "pattern": "[0-9]{5}(-[0-9]{4})?" } }
},
"else": {
  "properties": { "postal_code": { "pattern": "[A-Z][0-9][A-Z] [0-9][A-Z][0-9]" } }
}
```

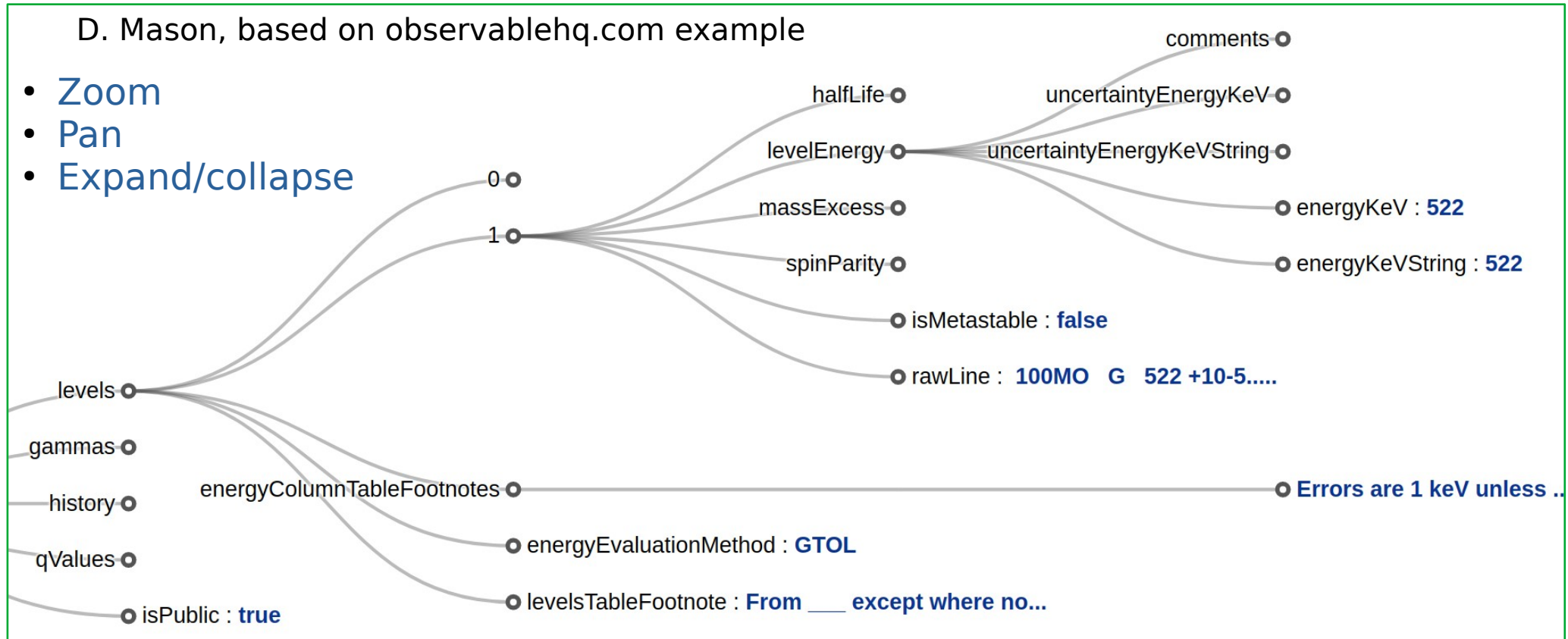
If / then / else logic

Visualization

- Greatly increases understanding among developers, users, evaluators
- Increases efficiency, supports group discussion

D. Mason, based on observablehq.com example

- Zoom
- Pan
- Expand/collapse



Summary

- New ENSDF data well-suited to & well-supported by JSON
 - A few reasons: hierarchical, heterogeneous, can update (subject to DB managers' approval) "on the fly"
 - JSON handles this data well, simplifies cross-references, e.g. want a parent to point to a daughter state:
nuclides["**255Lr**"]["**adopted**"]["**levels**"]["**3**"]
- Chose CouchDB
 - Object-oriented: database "understands" the data
 - Efficiency: "views" very important
 - License (compare with MongoDB license)
- Validation methods depend on the storage data structure
JSONSchema for JSON data. *I can share more on this & Dave Brown has experience with JSONSchema.*
- Side note: developing visualization early helps the group to understand data & structure and sidestep issues with unfamiliar formats during content discussions.

END