
Brookhaven
National Laboratory
U.S. DEPARTMENT OF
ENERGY
ENSDF Modernization
Donnie Mason
National Nuclear Data Center
@BrookhavenLab
11/13/2024

Evaluated Nuclear Structure Data File

ENSDF contains recommended nuclear structure and decay data for all the known nuclides

Includes:

- Nuclear level properties
- Gamma ray information
- Nuclear radiation and decay data

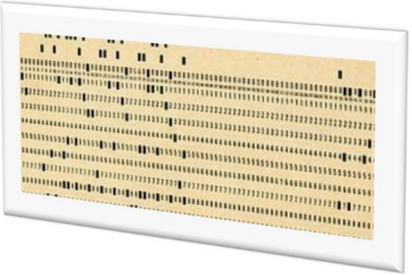
Legacy Format

SUMMARY
ENSDF STANDARD ONE-CARD FORMATS

RECORD TYPE	1–5	6	7	8	9	10–19	20–21	22–29	30–31	32–39	40–41	42–49	50–55	56–64	65	66–74	75–76	77	78–79	80
IDENTIFICATION	NUCID					DSID						DSREF				PUB	DATE			
GENERAL COMMENT	NUCID	§	+							CTEXT										
FLAGGED COMMENT	NUCID	§	+	•		SYM(FLAG)				CTEXT										
NORMALIZATION	NUCID			N		NR	DNR	NT	DNT	BR	DBR	NB	DNB							
PARENT	NUCID			P		E	DE	J				T	DT			QP	DQP			
Q-VALUE	NUCID			Q		Q⁻	DQ	SN	DSN	SP	DSP	QA	DQA			QREF				
LEVEL	NUCID	§		L		E	DE	J				T	DT	L		S	DS	C	MS	Q
GAMMA	NUCID	§		G		E	DE	RI	DRI	M		MR	DMR	CC	DCC	TI	DTI	C	†	Q
BETA	NUCID	§		B		E	DE	IB	DIB			LOG FT	DFT					C	UN	Q
EC	NUCID	§		E		E	DE	IB	DIB	IE	DIE	LOG FT	DFT			TI	DTI	C	UN	Q
ALPHA	NUCID	§		A		E	DE	IA	DIA	HF	DHF							C		Q
REFERENCE	AAA			R		KEYNUM				REFERENCE										
PARTICLE	NUCID			D	‡	E	DE	IP	DIP	ED		T	DT	L				C	†	Q
XREF	NUCID			X	@	DSID														

• = RTYPE
§ = COLUMN 6 IS BLANK OR 1 FOR THE FIRST CARD RECORD, ANY OTHER CHARACTER FOR CONTINUATION
† = C OR ? FOR COINCIDENCE
‡ = PARTICLE SYMBOL
+ = "C" OR "D" OR "T"
AAA = MASS NUMBER
@ = ANY CHARACTER

80 Column ASCII format



```
137CS PN                                                                    6
137CS  L 0.0           7/2+                  30.08 Y   9                   A
137CSX L XREF=ACDEFGH
137CS2 L %B-=100$MOMM1=+2.8413 1 (1989Ra17)$MOME2=+0.051 1 (1989Ra17)
137CS cL T$Deduced by evaluators using the Limitation of Relative Statistical
137CS2cL Weights (LRSW) method for analyzing the following set of
137CS3cL discrepant (|h{+2}/|n=18.6) experimental values: 10970 d {I20}
137CS4cL (2004Sc04); 11018 d {I10} (2002Un02); 10941 d {I7} (1992Go24);
137CS5cL 10968 d {I5} (1990Ma15); 11009 d {I11} (1980Ho17); 10906 d {I33}
137CS6cL (1978Gr08); 11034 d {I29} (1973Co39); 11021 d {I5} (1973Di01); 11023 d
137CS7cL {I37} (1972Em01); 10921 d {I17} (1970Wa19); 1
137CS8cL 11286 d {I256}, 10921 d {I183} (1965Fl01); 11
```

80 column fields

"Continuation record"

A LOT of data stored in comments
Non-standardized entry

Brookhaven National Laboratory

JSON Schema

What is JSON Schema?

- Declarative language for defining JSON data structure and validation rules.
- Ensures data conforms to specified format.
- Improves data integrity, documentation, and automation.

Key Features

- **Validation**: Check data types, required fields, range, and format.
- **Documentation**: Describes expected JSON structure clearly.
- **Interoperability**: Facilitates data exchange between systems.
- **Nested Structures**: Supports objects and arrays with constraints.
- **Reuse**: Allows schema component reuse via `$ref`.

JSON Schema is the vocabulary that enables JSON data consistency, validity, and interoperability at scale.

JSON Schema

Structure of JSON Schema

- Root object with `type`, `properties`, `required`, etc.
- Defines types for each property (`string`, `integer`, `boolean`).
- Specifies constraints like `minimum`, `maximum`, `pattern`.

Common Keywords

- `type`: Specifies data type (`string`, `integer`, etc.).
- `properties`: Defines attributes of an object.
- `required`: Lists mandatory properties.
- `enum`: Restricts values to a predefined set.
- `pattern`: Defines regex for string matching.

Use Cases

- **API Interfaces**: Define request/response formats.
- **Data Storage**: Validate data in NoSQL database.
- **Data Validation**: Validate data before processing.

Example: Adult

```
{
    "type": "object",
    "properties": {
      "name": { "type": "string" },
      "age": { "type": "integer", "minimum": 18 }
    },
    "required": ["name", "age"]
  }
```

Pass

```
{
    "name": "John Doe",
    "age": 25
}
```

Fail

```
{
    "name": {
        "first": "John",
        "last": "Doe"
    },
    "age": 25
}
```

```
{
    "name": "John Doe",
    "age": 15
}
```

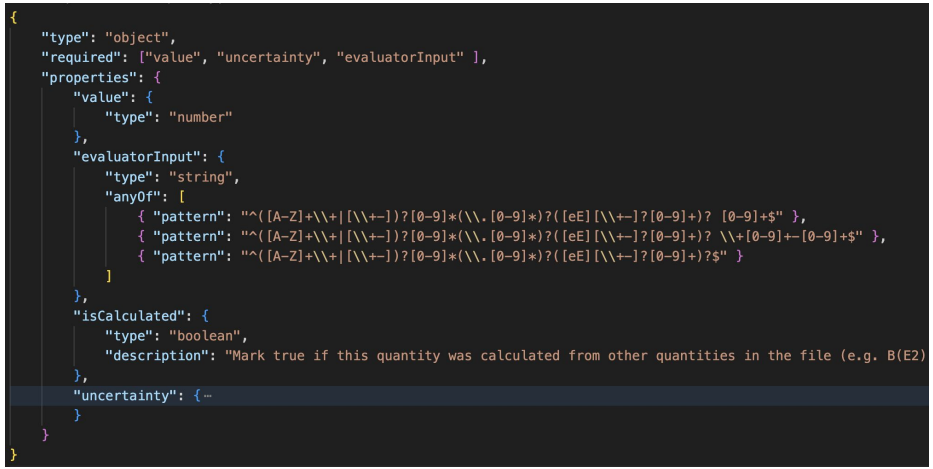
Halflife Schema

- **No Unevaluated Properties**
  - Ensures that only the properties listed in the schema are allowed.
- **Quantity**
  - $ref to reuse schema components
  - References quantity.json schema for validation
- **Properties**
  - “comments”
    - References basic-comments.json schema for validation
  - “unit”
    - Enumerated string
    - A predefined set of valid units (e.g., “h”, “m”, “s”)
    - Ensures the value matches one of the valid unit options.
  - “measurements”
    - References measurements.json schema for validation
- **Requires the “unit” property**
  - Halflife quantities given without a unit are invalid

```
{
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "$id": "file://local.file/components/halflife.json",
    "unevaluatedProperties": false,
    "$ref": "quantity.json",
    "properties": {
        "comments": {
            "$ref": "basic-comments.json"
        },
        "unit": {
            "type": "string",
            "enum": ["Gy","My","ky","y","d","h","m","s","ms","us","ns","ps",
                     "fs","as","zs"]
        },
        "measurements": {
            "$ref": "measurements.json"
        }
    },
    "required": [ "unit" ]
}
```

Quantity Schema

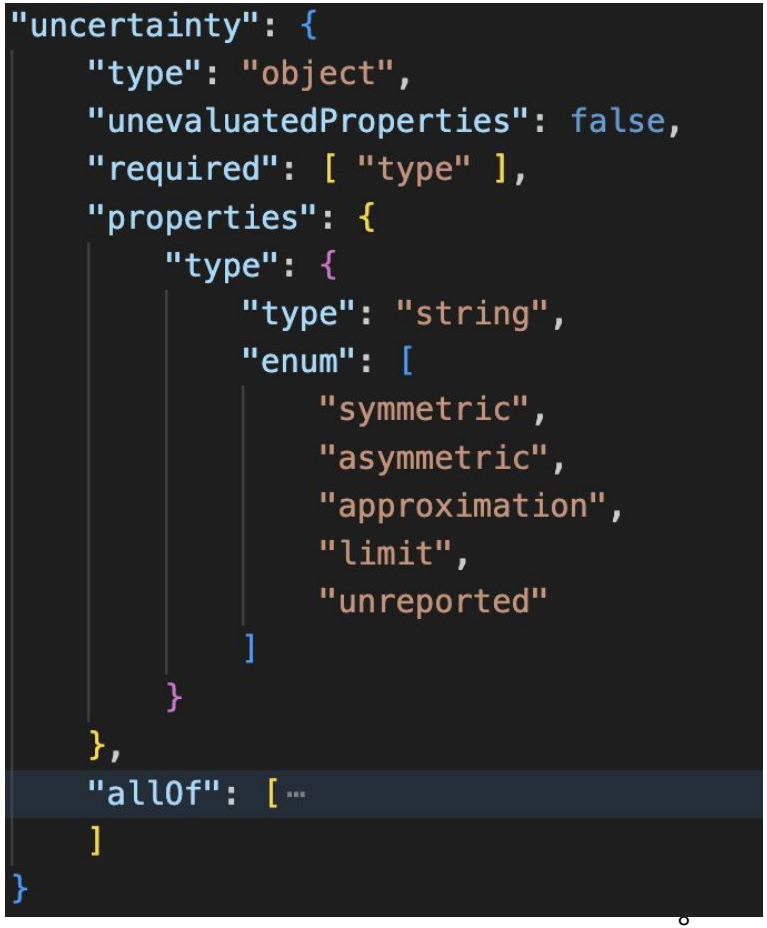
- **Required properties**
  - “value”
  - “uncertainty”
  - “evaluatorInput”
- **Properties**
  - “value”: number (eg. 1, 2.34, 1e-4, etc.)
  - “evaluatorInput”
    - String: validated against any of the following regex patterns
  - “isCalculated”
    - Boolean flag
  - “Uncertainty”
    - Object…



Quantity Schema

Uncertainty property

- Requires the property type
- Type
  - Enumerated string specifying the uncertainty type
  - “symmetric”, “asymmetric”, “unreported” etc.



Quantity Schema

Uncertainty property

- Requires the property type
- Type
  - Enumerated string specifying the uncertainty type
  - “symmetric”, “asymmetric”, “unreported” etc.
- Conditional validation “allOf”
  - If type == “symmetric”
    - Requires
      - “value” (number)

Quantity Schema

Uncertainty property

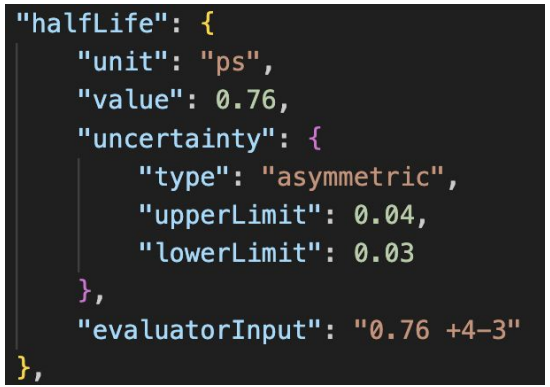
- Requires the property type
- Type
  - Enumerated string specifying the uncertainty type
  - “symmetric”, “asymmetric”, “unreported” etc.
- Conditional validation “allOf”
  - If type == “symmetric”
    - Requires
      - “value” (number)
  - If type == “asymmetric”
    - Requires
      - “upperLimit” (number)
      - “lowerLimit” (number)

Quantity Schema

Uncertainty property

- Requires the property type
- Type
  - Enumerated string specifying the uncertainty type
  - “symmetric”, “asymmetric”, “unreported” etc.
- Conditional validation “allOf”
  - If type == “symmetric”
    - Requires
      - “value” (number)
  - If type == “asymmetric”
    - Requires
      - “upperLimit” (number)
      - “lowerLimit” (number)
  - If type == “limit”
    - Requires
      - “isInclusive” (boolean)
      - “limitType” (enumerated string)

Halflife Examples



ENSDF API

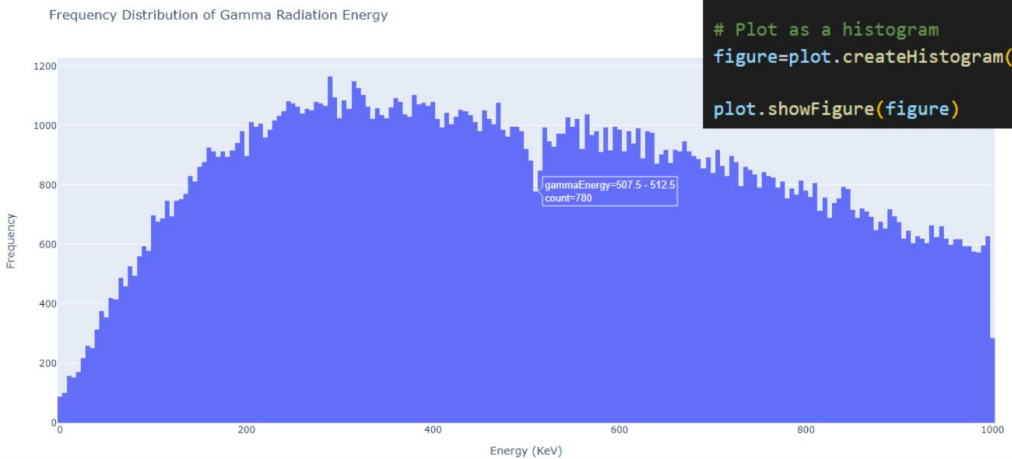
```python
# Initialize API
api = ensdfAPI(ipAddress="127.0.0.1", port=5001)

# Get all gammas 0-1000 keV
values_dict=api.filterByGammas(0,1000)
dataframe = plot.createViewDataFrame(values_dict)

# Label plot
plot.configuration.setAxisTitle("x", "Energy (KeV)")
plot.configuration.setAxisTitle("y", "Frequency")
plot.configuration.setTitle("Frequency Distribution of Gamma Radiation Energy")

# Plot as a histogram
figure=plot.createHistogram(dataframe, "gammaEnergy")

plot.showFigure(figure)
```



Brookhaven National Laboratory

ENSDF Editor

Electron for Desktop Application:

- Leverages **web technologies** (HTML, CSS, JavaScript) for cross-platform desktop apps.
- Combines **Node.js** backend with a **Chromium** frontend to provide a rich UI experience.
- **Open source project** maintained by the OpenJS Foundation

JSON Editing Interface:

- Provides a **user-friendly interface** to create and modify JSON data.
- Knowledge of the schema and the expected format for each field is not needed

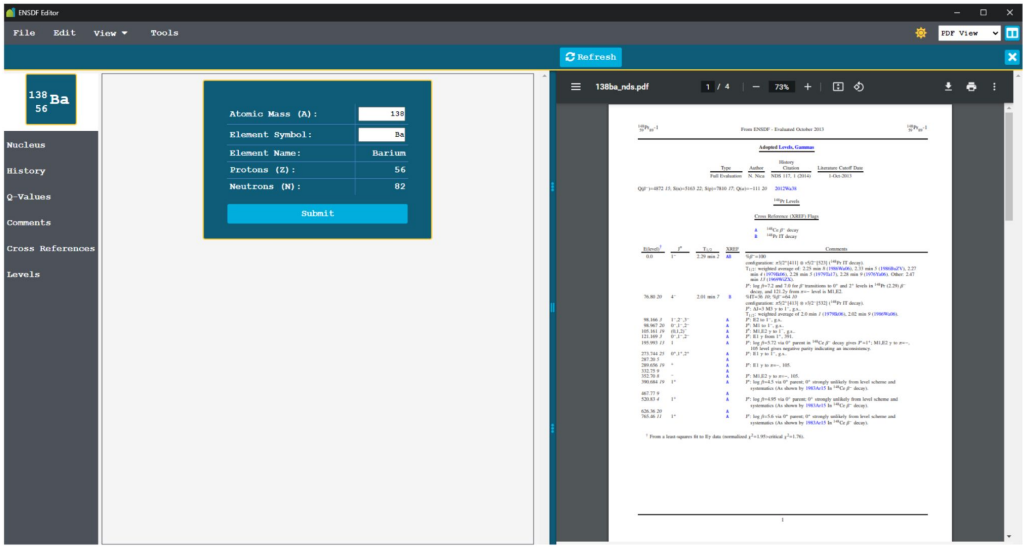
Customized UI Components:

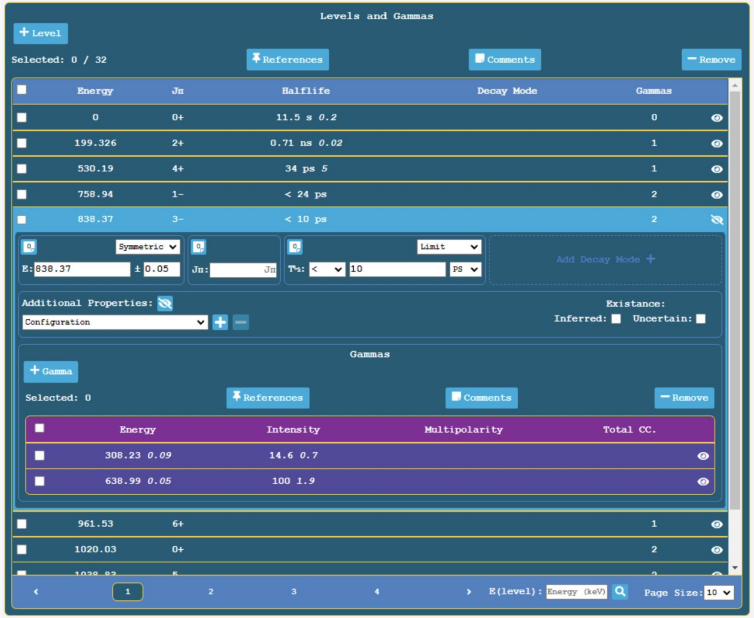
- Interactive controls for different data types (e.g., levels, quantities, comments, measurements).

Cross-Platform:

- Works on major platforms (Windows, macOS, Linux) through Electron
- Allows evaluators to create datasets in any environment.

ENSDF Editor





Brookhaven National Laboratory

JSON C++ nlohmann/json

- Link: https://github.com/nlohmann/json
- **Header-only**: No need for external libraries or complex setup—just include a single header file.
- **Simple, Intuitive API**: Modern C++11+ syntax for easy parsing, serialization, and manipulation of JSON data.
- **STL Compatibility**: Seamless integration with `std::vector`, `std::map`, and other C++ containers for JSON handling.
- **Automatic Type Conversion**: Effortlessly convert between C++ types (e.g., `int`, `std::string`) and JSON.
- **Custom Serialization**: Easily define custom serialization logic for your own types using `to_json` and `from_json` functions.

```
Total time taken to process 3438 files: 69.0247 seconds
Total data processed: 571.773 MB
Average speed: 8.2836 MB/s
```

JSON C++

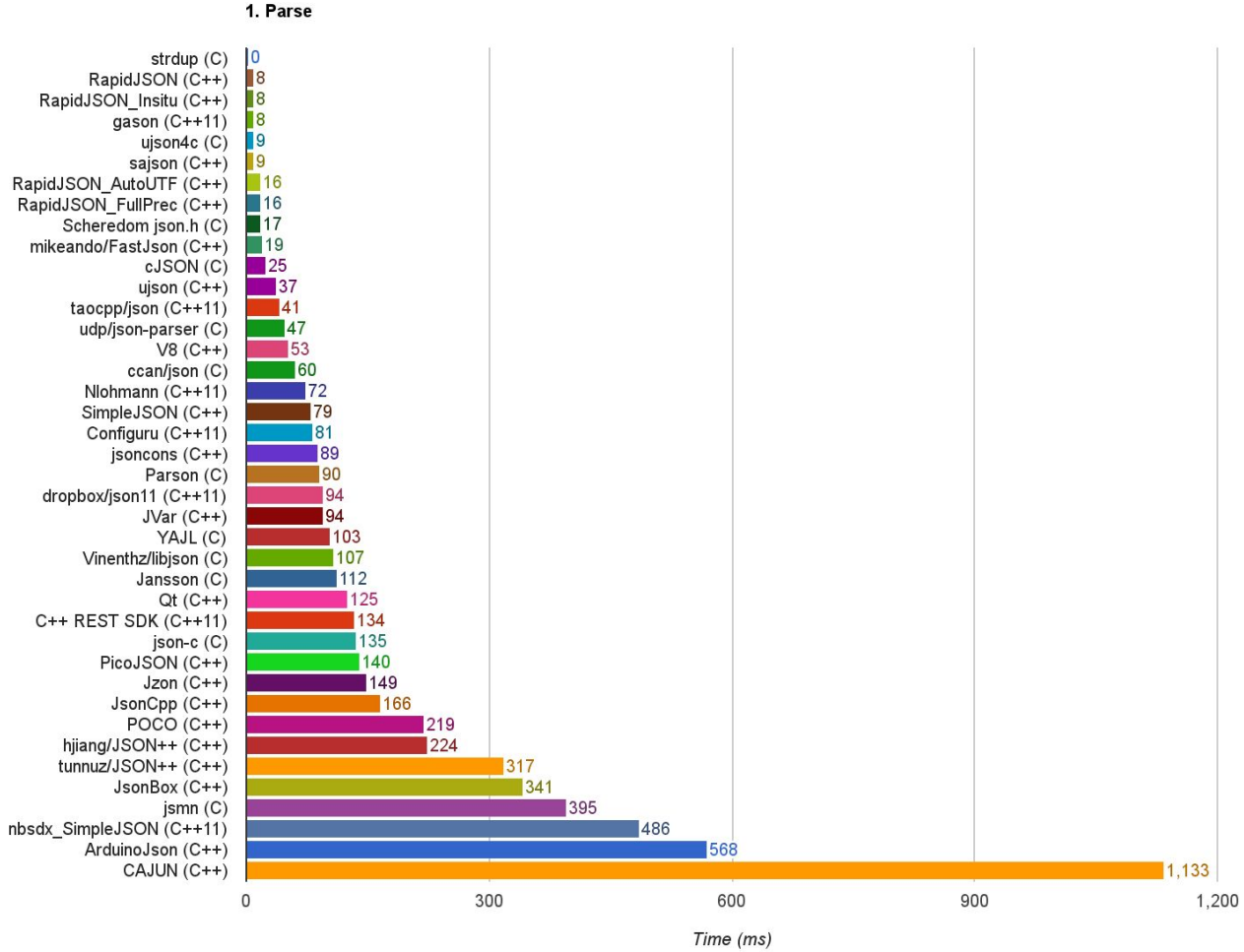
RapidJSON

- Link: https://rapidjson.org/
- RapidJSON is **small** but **complete**. It supports both SAX and DOM style API. The SAX parser is only a half thousand lines of code.
- RapidJSON is **fast**. Its performance can be comparable to `strlen()`. It also optionally supports SSE2/SSE4.2 for acceleration.
- RapidJSON is **self-contained** and **header-only**. It does not depend on external libraries such as BOOST. It even does not depend on STL.
- RapidJSON is **memory-friendly**. Each JSON value occupies exactly 16 bytes for most 32/64-bit machines (excluding text string). By default it uses a fast memory allocator, and the parser allocates memory compactly during parsing.
- RapidJSON is **Unicode-friendly**. It supports UTF-8, UTF-16, UTF-32 (LE & BE), and their detection, validation and transcoding internally. For example, you can read a UTF-8 file and let RapidJSON transcode the JSON strings into UTF-16 in the DOM. It also supports surrogates and "\u0000" (null character).

```
Total time taken to process 3438 files: 22.518 seconds
Total data processed: 571.773 MB
Average speed: 25.3919 MB/s
```

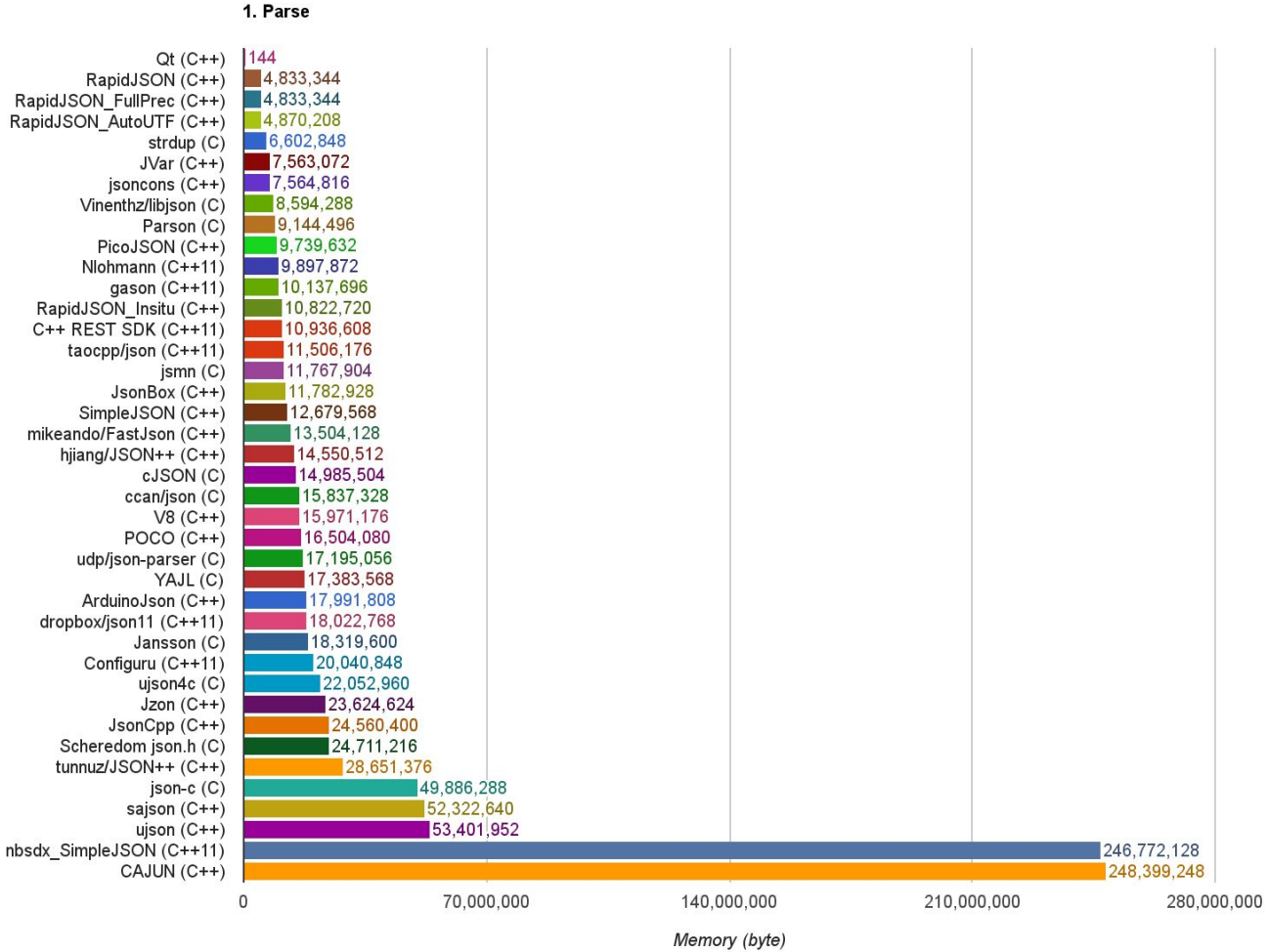
JSON C++ Benchmarks

https://github.com/miloyip/nativejson-benchmark#parsing-time



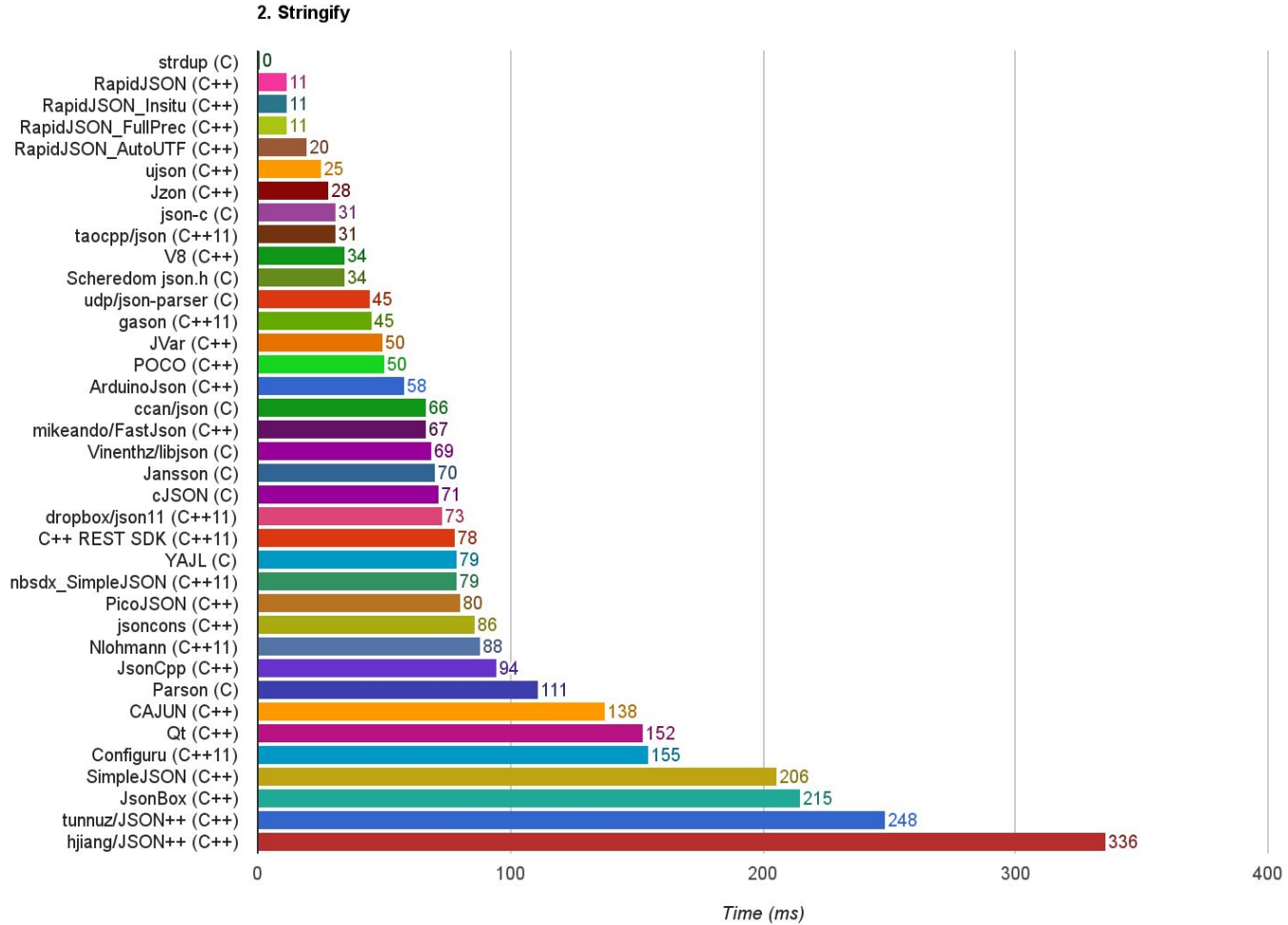
JSON C++ Benchmarks

https://github.com/miloyip/nativejson-benchmark#parsing-time



JSON C++ Benchmarks

https://github.com/miloyip/nativejson-benchmark#parsing-time



Demonstration