

Simpler is Better: JDT as a New Schema Language for JSON Documents

Benedikt Sennebogen¹, Antoon Bronselaer², Toon Boeckling², Uta Störl³ and Meike Klettke¹

¹University of Regensburg, Regensburg, Germany

²Department of Telecommunications and Information Processing, Ghent University, Ghent, Belgium

³FernUniversität in Hagen, Hagen, Germany

Abstract

In computer science, simple, easy-to-learn methods and complex, powerful solutions are often simultaneously available. The simple solutions are often used when interacting with users. The currently available JSON schema languages are highly expressive, but not easy to write manually. Therefore, we propose a very intuitive yet expressive language, called JSON Document Type (JDT). In addition to features for defining structural constraints, simplicity, similarities to the English language, and compactness have been key design criteria. In this demo, we showcase the JDT language and two different tools: an editor that can be used to easily define a JDT schema and a tool to validate JSON documents against JDT. The target audience are researchers with different backgrounds looking for an intuitive method to define the structure of their JSON documents to streamline their data processing applications.

Tool URL: <https://go.ur.de/jdt>

Video URL: <https://www.youtube.com/watch?v=AAdzBW3cMGw>

Keywords

JSON, schema language, schema constraints, schema editor, validation

1. Introduction

In recent years, various data formats (such as XML, JSON, and TOON) have been developed, all of which allow for greater flexibility than relational-style data. These formats initially started as schema-less formats, but the need for schema languages quickly emerged from tasks like query optimization and data validation.

All schema languages can be used in either a schema-first or schema-second approach. In a schema-first approach, users define the schema in advance. In a schema-second approach, the initial schema is derived from sample data; afterwards it is reviewed and adapted by users. In both cases, it is therefore necessary that users can easily understand and define schemas. There is thus a conflicting goal between the desired expressiveness of the schema language and the desired simplicity and, consequently, ease of use of the schema language.

In this demo paper, we introduce a new schema language and validator for JSON documents called *JSON Document Type* (JDT) Schema Language. Its main advantage is *simplicity by design*, advocated by recent findings on the high complexity of modern JSON schema [1, 2].

ER2026: Companion Proceedings of the 45th International Conference on Conceptual Modeling: Doctoral Consortium, Tutorials, Posters and Demos, October 5-8, 2026, St. John's, NL, Canada

✉ benedikt.sennebogen@stud.uni-regensburg.de (B. Sennebogen); antoon.bronselaer@ugent.be (A. Bronselaer); toon.boeckling@ugent.be (T. Boeckling); uta.stoerl@fernuni-hagen.de (U. Störl); meike.klettke@ur.de (M. Klettke)

🌐 www.uni-regensburg.de/en/informatics-data-science/faculty/facilities/data-engineering (B. Sennebogen); <https://ddcm.ugent.be/> (A. Bronselaer); <https://ddcm.ugent.be/> (T. Boeckling); <https://www.fernuni-hagen.de/dbis/> (U. Störl); www.uni-regensburg.de/en/informatics-data-science/faculty/facilities/data-engineering (M. Klettke)

🆔 0009-0005-8458-3411 (B. Sennebogen); 0000-0001-6663-192X (A. Bronselaer); 0000-0002-4972-5294 (T. Boeckling); 0000-0003-2771-142X (U. Störl); 0000-0003-0551-8389 (M. Klettke)



© 2026 Copyright © 2026 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

2. Related Work

Schema languages have been extensively investigated in the case of XML [3, 4, 5] and their differences in *expressiveness* are well understood by mapping them onto *Regular Tree Grammars* (RTGs) [6]. The machinery of RTGs shows that some XML schema languages are less expressive¹ than others, leading to a hierarchy of schema languages. Typically, lower expressivity comes with simpler validation algorithms [6]. For XML documents, DTDs are a well-known simple schema language [3] whereas XML schema [4] and RelaxNG [5] are more expressive.

Schema languages for JSON data [7] and validation approaches are a much more recent topic of research. In contrast to the case of XML, the variety of schema languages for JSON data is much smaller, with JSON Schema being the main object of study [8, 9]. However, it has been argued that the latest version of JSON Schema (i.e., “Modern” JSON Schema) introduces features that make it hard to understand [1]. Empirical studies show that many of these advanced features have low prevalence in real-life schema instances [2, 10, 11]. One way of dealing with this is to develop specific tools to aid in understanding JSON Schema validation [12]. This solves only part of the problem, since a higher number of concepts in a schema language leads to more time to answer questions about them and makes schemas harder to create from the analyst’s point of view [13].

Therefore, we argue that schema languages for JSON with a conceptually simplified structure can fill a void in the current landscape of JSON schema languages. The JDT language proposed in this paper aims to do precisely this. Similar to DTD and RelaxNG for XML and JSON Type Definition [14] for JSON, a JDT instance is in itself not a JSON document. This is a deliberate choice: we choose a concise notation to improve readability.

3. JDT – JSON Document Type

The JSON Document Type language (or JDT for short) is designed as an expressive, yet highly readable and accessible language. The main design principles have been to maximize the understandability of the language and the user-friendliness. To achieve this, we minimized the use of special characters and defined properties in a ‘Pythonic’ way. For reduced verbosity, we’ve eliminated the use of brackets, utilized indentation, and minimized the need for quotation marks. The core concept of JDT focuses on defining the *structural constraints* that JSON documents must satisfy to be considered valid. Therefore, the language includes the following constraints:

1. Root constraints at the beginning of a schema allow users to define ownership and version.
2. Global constraints allow restrictions for the entire document. For example, users can restrict undefined keywords to not be valid or require the schema to be an object.
3. Simple constraints define data types or certain values, like `string`, `number`, or `null`.
4. Value, length and occurrence constraints, like `minimum 1`, `longer 5`, or `required` restrict the data value and its required or optional existence.
5. Regular expressions can be defined inside a match function: `match("regex")`. Arrays can be defined in the schema with an array function, e.g.: `array(datatype)`.

Moreover, constraints can be combined with Boolean connective operators (and, or, and not).

¹Here, expressiveness relates only to which structures can be modeled.

In Listing 1, we introduce a *JDT definition*. It requires an object with a container named `user` consisting of two keys: `id` defined as `number` and `name` defined as `string`. Both keys are required in all valid user objects. In Listing 2 is a *JSON document* fulfilling the JDT definition.

```
is type object
user:
  id is number, required
  name is string, required
```

Listing 1: JDT Schema for a simple application

```
{  "user": {
    "id": 42,
    "name": "Leibniz"
  } }
```

Listing 2: Valid JSON document

We expand our schema definition a bit (see Listing 3), using many of the constraints presented on the previous page, and show in Listing 4 a *JSON Schema* that defines the same constraints.

```
is type object
user:
  is closed
  id is number, required, not null
  name is required, not boolean
  age is optional, (number or null)
  hobbies is array(string), required
  eyecolor is "blue" or "green" or "brown"
```

Listing 3: JDT Schema, extended

The clearest difference between JDT and JSON Schema is not only the document length, but rather the simplicity of JDT. There are no curly braces, almost no colons or quotation marks, and the use of commas is limited. This enhances both usability and readability of JDT, especially for non-expert users.

```
{ "type": "object",
  "properties": {
    "id": {
      "type": "number" },
    "name": {
      "not": {
        "type": "boolean" } },
    "age": {
      "type": ["number", "null" ] },
    "hobbies": {
      "type": "array",
      "items": {
        "type": "string" } },
    "eyecolor": {
      "type": "string",
      "enum": ["blue", "green", "brown" ] },
    "required": ["id", "name", "hobbies"],
    "additionalProperties": false }
```

Listing 4: Corresponding JSON Schema

4. JDT Ecosystem

We developed two JDT-related tools: a *JDT editor* that can be used to easily define JDT constraints, and a *JSON document validation* tool. Both tools are easily accessible via a universal *JDT web interface*. In addition, an *Application Programming Interface (API)* is provided to interact with our framework as well. Both can be found here: <https://go.ur.de/jdt>

JDT editor: Context-aware, auto-complete extensions increase writing flow. A tight IDE integration, especially for the open-source Visual Studio Code platform, enables *AI auto-complete suggestions* with GitHub Copilot. Coupled with its readability, JDT is designed to be ready for an AI future where developers do less writing and more proofreading.

JSON document validation: The program uses a lexer to tokenize the schema. A parser then recursively descends through the token stream with a lookahead function to create an *Abstract Syntax Tree (AST)*. Afterwards, the validator then matches the constraints in the AST against a JSON file. In the validation process, clear error messages tell the user which constraint could not be validated to enhance the debugging experience (see Figure 1). If the document can be validated, a success message appears.

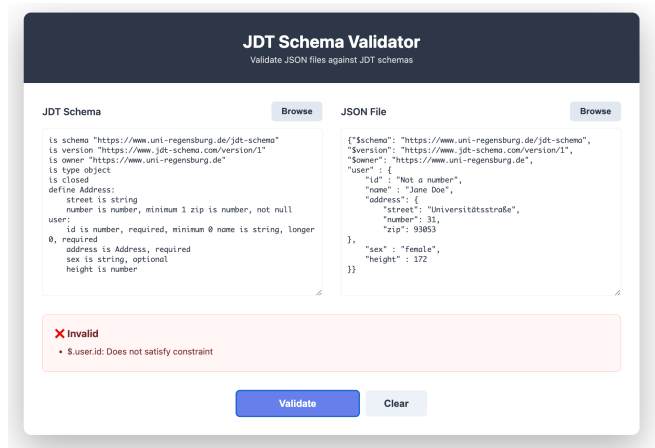


Figure 1: The web interface for JDT Schema definition and JSON document validation.

5. Evaluation

We have evaluated the language and both tools according to different characteristics.

Data set: We generated samples with 10 to 5000 user entities, each with 4 properties.

Schema size: JDT requires far fewer tokens than JSON Schema. In Figures 3 and 4, the JSON Schema has 137 tokens, while the JDT document has only 63 tokens (less than half).²

Run-time: An initial performance comparison of the validation against the Python library of JSON Schema showed that for startup, JDT is up to 20% faster. For validation, JDT is about 3.1 times faster. Startup time is linear in $O(n)$ with the schema’s number of characters n . With respect to the file size, the main result is that JDT validates in linear time $O(m)$ with m being the number of characters in the JSON file. While Rust-based implementations like *jsonschema_rs* are significantly faster, the native Python library remains the most well-known library, resulting in speed gains for many users. The *fastjsonschema* library was faster at validation, but slower at startup.

Table 1 shows the performance comparison between JSON Schema and JDT in the validation process using the JSON Schema Python Library on an Apple MacBook Pro 2023 with M2 Pro and 16 GB. The batch validation of user arrays was averaged over 100 iterations with cumulative values, excluding startup.

Size	JDT	JSON Schema	Ratio
10	0.0456 ms	0.1516 ms	3.3x
100	0.4548 ms	1.4385 ms	3.2x
1,000	4.4803 ms	13.8986 ms	3.1x
5,000	22.1947 ms	69.6587 ms	3.1x

Table 1: Comparison of the validation speed

6. Demo Outline

It was a classic joke among developers: while learning the basic syntax of XML just takes you 10 minutes, mastering the ecosystem of XML technologies takes a lifetime. Learning JSON syntax is also not complicated. Our goal is to offer a JSON schema language as part of the JSON ecosystem that can also be learned in 10 minutes. In this demo, we will present the JDT language and its auto-complete editor. For the demo, we prepare datasets from different domains, but will also offer the opportunity to develop a JDT schema for their own JSON data. After defining the schema, you can validate your data to detect syntax errors, find data quality problems and detect hidden heterogeneity in your data. You’re also welcome to discuss with us how the language should be evolved.

²This not only means that JDT is less token-consumptive, which has an effort onto the runtime, but also reduces visual clutter and mental overhead resulting in better usability of the schema language.

7. Conclusion and Future Work

With JDT, we propose an understandable schema language for users both within and outside the field of computer science. In some ways, its compactness is similar to DTDs, which were developed for XML documents. In contrast to DTDs, JDT includes basic data types, since their lack in DTDs was one of the main criticisms of that schema language.

There are only *two semantic differences* between JDT and JSON Schema: References cannot be defined; this design decision was made to allow validation of each JSON document as a separate file. Furthermore, *conditional constraints* have not yet been included.

Our next step will be an *empirical study* with various user groups to compare the understanding, creation, and use of JDT and JSON Schema. The question of whether conditional constraints are to be added will also be a topic of the empirical study.

Declaration on Generative AI

During development, authors of this work have deployed AI in coding and spell-checking. The authors have reviewed and edited the content and take full responsibility for the published article.

References

- [1] L. Attouche, M. Baazizi, D. Colazzo, G. Ghelli, C. Sartiani, S. Scherzinger, Validation of Modern JSON Schema: Formalization and Complexity, *Proc. ACM Program. Lang.* 8 (2024) 1451–1481.
- [2] L. Attouche, M. Baazizi, D. Colazzo, G. Ghelli, S. Klessinger, C. Sartiani, S. Scherzinger, Elimination of annotation dependencies in validation for Modern JSON Schema, *Theor. C. S.* 1063 (2026) 115645.
- [3] T. Bray, J. Paoli, S. Sperberg-McQueen, E. Maler, F. Yergeau, Extensible Markup Language (XML) 1.0, 2001. URL: <https://www.w3.org/TR/xml/#dt-doctype>.
- [4] H. Thompson, D. Beech, M. Moloney, N. Mendelsohn, XML Schema Part 1: Structures. W3C Recommendation., 2001. URL: <http://www.w3.org/TR/xmlschema-1/>.
- [5] J. Clark, M. Murata, RELAX NG Specification, 2001. URL: <http://www.oasis-open.org/committees/relax-ng/spec-20011203.html>.
- [6] M. Murata, D. Lee, M. Mani, K. Kawaguchi, Taxonomy of XML schema languages using formal language theory, *ACM Trans. Internet Techn.* 5 (2005) 660–704.
- [7] P. Bourhis, J. L. Reutter, F. Suárez, D. Vrgoc, JSON: Data model, Query languages and Schema specification, in: *PODS, ACM*, 2017, pp. 123–135.
- [8] JSON Schema, 2026. URL: <https://json-schema.org/>.
- [9] F. Pezoa, J. L. Reutter, F. Suárez, M. Ugarte, D. Vrgoč, Foundations of JSON Schema, in: *Proc. Int. Conf. World Wide Web*, 2016, pp. 263–273. doi:10.1145/2872427.2883029.
- [10] M. Baazizi, D. Colazzo, G. Ghelli, C. Sartiani, S. Scherzinger, An Empirical Study on the "Usage of Not" in Real-World JSON Schema Documents, in: *ER*, volume 13011 of *LNCS*, 2021, pp. 102–112.
- [11] A. Alsulami, Empirical Analysis of JSON Schema Use, Master's thesis, Rochester Institute of Technology, 2021. MSc capstone, Data Unity Lab.
- [12] L. Attouche, M. Baazizi, D. Colazzo, G. Ghelli, K. C. Imo, S. Klessinger, C. Sartiani, S. Scherzinger, JTutor: JSON Schema Validation Explained, in: *DBPL, ACM*, 2025, pp. 1–6.
- [13] A. Bajaj, The effect of the number of concepts on the readability of schemas: an empirical study with data models, *Requir. Eng.* 9 (2004) 261–270.
- [14] U. Carion, JSON Type Definition, 2020. URL: <https://datatracker.ietf.org/doc/rfc8927/>.