JSON

Péter Jeszenszky

October 4, 2024

JSON (JavaScript Object Notation)

- Lightweight, textual, and platform independent data exchange format.
- Used for representing structured data.
- Can be read and written easily by humans.
- Can be generated and processed easily by computer programs.
- Originates from the ECMAScript programming language.
- Website: https://www.json.org/

ECMAScript

- ECMAScript is standardized JavaScript.
- The current version is the 15th edition:
  - Ecma International, *ECMAScript 2024 Language Specification*, Standard ECMA-262, 15th ed., June 2024. https://www.ecma-international.org/publications-and-standards/standards/ecma-262/
- The next version currently under development is ECMAScript 2025:
  - *ECMAScript 2025 Language Specification* https://tc39.es/ecma262/

JavaScript

- The term JavaScript is used for the implementations of ECMAScript by different vendors.
  - See also: *JavaScript technologies overview* https://developer.mozilla.org/en-US/docs/Web/JavaScript/JavaScript_technologies_overview

JavaScript Engines (1)

- SpiderMonkey (written in: C/C++; license: Mozilla Public License 2.0) https://spidermonkey.dev/
  - The JavaScript engine of the Mozilla Project.
- V8 (written in: C++; license: New BSD License) https://v8.dev/ https://github.com/v8/v8/
  - The JavaScript engine of Chromium.
- JavaScriptCore (written in: C++; license: LGPLv2) https://developer.apple.com/documentation/javascriptcore https://github.com/WebKit/WebKit/tree/main/Source/JavaScriptCore
  - The JavaScript engine developed for the WebKit rendering engine.

JavaScript Engines (2)

- GraalVM Community Edition (CE) (written in: Java; license: GPLv2)
  https://www.graalvm.org/ https://github.com/oracle/graal
  - Available on Linux, macOS, and Windows systems.
  - See: https://www.graalvm.org/latest/reference-manual/js/
- Hermes (written in: C++; license: MIT License)
  https://hermesengine.dev/ https://github.com/facebook/hermes
  - JavaScript engine optimized to run React Native applications.

JavaScript Engines (3)

- JerryScript (written in: C; license: Apache License 2.0)
  https://jerryscript.net/
  https://github.com/jerryscript-project/jerryscript
  - Lightweight JavaScript engine for IoT devices.

Node.js (1)

- A JavaScript runtime environment built on the V8 JavaScript engine that is designed to build scalable network applications.
- Website: https://nodejs.org/ https://github.com/nodejs/node
- License: MIT License
- Written in: C++, JavaScript
- Platform: Linux, macOS, Windows

Node.js (2)

- Lets the developers to build applications in JavaScript that run outside a web browser.
- Can be used for client-side and server-side application development.
- Its package ecosystem, npm, is the largest ecosystem of open source libraries in the world.

Node.js (3)

"Hello, World!" example:

- Source: https://nodejs.org/en/docs/guides/getting-started-guide/

```
const { createServer } = require('node:http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello, World!\n');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Node.js (4)

Node.js frameworks:

- Fastify (license: MIT License) https://fastify.dev/ https://github.com/fastify/fastify
- Express (license: MIT License) http://expressjs.com/ https://github.com/expressjs/express
- Koa Express (license: MIT License) https://github.com/koajs/koa https://github.com/koajs/koa
- Meteor (license: MIT License) https://www.meteor.com/ https://github.com/meteor/meteor
- NestJS (license: MIT License) https://nestjs.com/ https://github.com/nestjs/nest
- Sails.js (license: MIT License) https://sailsjs.com/ https://github.com/balderdashy/sails
- ...

Compatibility

- ECMAScript compatibility of implementations:
  - *ECMAScript Compatibility Tables*
    https://compat-table.github.io/compat-table/es6/
    https://github.com/compat-table/compat-table

ECMA International

- International non-profit standards organization.
- Target domains: information and communication technology (ICT), consumer electronics (CE)
- Was founded in 1961 originally, operates under its current name since 1994.
  - European Computer Manufacturers Association (ECMA)
- Website: https://www.ecma-international.org/

History

- Discovered and popularized by Douglas Crockford.
- Was "discovered" in 2001.
- Crockford originally used JSON for communication between JavaScript clients and Java servers.
- IEEE Computer Society, *Discovering JavaScript Object Notation with Douglas Crockford*, 28 March 2012. https://www.youtube.com/watch?v=kc8BAR7SHJI
  - "I don't claim to have invented it, because it already existed in nature. I just saw it, recognized the value of it, gave it a name, and a description, and showed its benefits. But I did not invent it. I don't claim to be the first person to have discovered it."
- Source: https://inkdroid.org/2012/04/30/lessons-of-json/

File Properties

- File extension: `.json`
- IANA media type: `application/json`

Specification

- Ecma International, *The JSON Data Interchange Format*, Second Edition, Standard ECMA-404, December 2017. https://www.ecma-international.org/publications-and-standards/standards/ecma-404/
- T. Bray (ed.). *RFC 8259: The JavaScript Object Notation (JSON) Data Interchange Format*. December 2017. https://www.rfc-editor.org/rfc/rfc8259
- ISO/IEC 21778:2017: *Information technology – The JSON data interchange syntax* https://www.iso.org/standard/71616.html

JSON vs. ECMAScript

- Starting with ECMAScript 2019, JSON is a syntactic subset of ECMAScript.
- See:
  - *Subsume JSON a.k.a. JSON ⊂ ECMAScript*, 14 August 2019. https://v8.dev/features/subsume-json
  - Axel Rauschmayer, *Exploring ES2018 and ES2019 – JSON superset*. https://exploringjs.com/es2018-es2019/ch_json-superset.html

JSON vs. XML (1)

- JSON can be used as an alternative to XML for data exchange.
- Provides the same advantages as XML but without its disadvantages.
- See: *JSON: The Fat-Free Alternative to XML*
  https://www.json.org/xml.html

JSON vs. XML (2)

Common characteristics of JSON and XML:

- Simplicity (undoubtedly JSON is the winner)
- Can be read and written easily by humans
- Can be generated and processed easily by computer programs (undoubtedly JSON is the winner)
- Interoperability
- Open standard
- Self-describing data representation
- Universal data exchange format

JSON vs. XML (3)

- The main difference is that JSON is data-oriented, while XML is document-oriented.
  - JSON is the perfect choice for representing data structures.
    - JSON is less verbose compared to XML.
  - Use XML for document-oriented applications.
    - XML is extensible and has a more mature infrastructure.

JSON vs. XML (4)

Example:

- XML:

```xml
<movie>
  <title>The Dark Knight</title>
  <year>2008</year>
  <url>https://www.imdb.com/title/tt0468569/</url>
  <standalone>false</standalone>
</movie>
```

- JSON:

```json
{
  "movie": {
    "title": "The Dark Knight",
    "year": 2008,
    "url": "https://www.imdb.com/title/tt0468569/",
    "standalone": false
  }
}
```

JSON vs. XML (5)

- XML:

```xml
<properties>
  <property name="user.home">/home/jeszy</property>
  <property name="user.name">jeszy</property>
</properties>
```

- JSON:

```json
{
  "properties": {
    "property": [
      {
        "@name": "user.home",
        "#text": "/home/jeszy"
      },
      {
        "@name": "user.name",
        "#text": "jeszy"
      }
    ]
  }
}
```

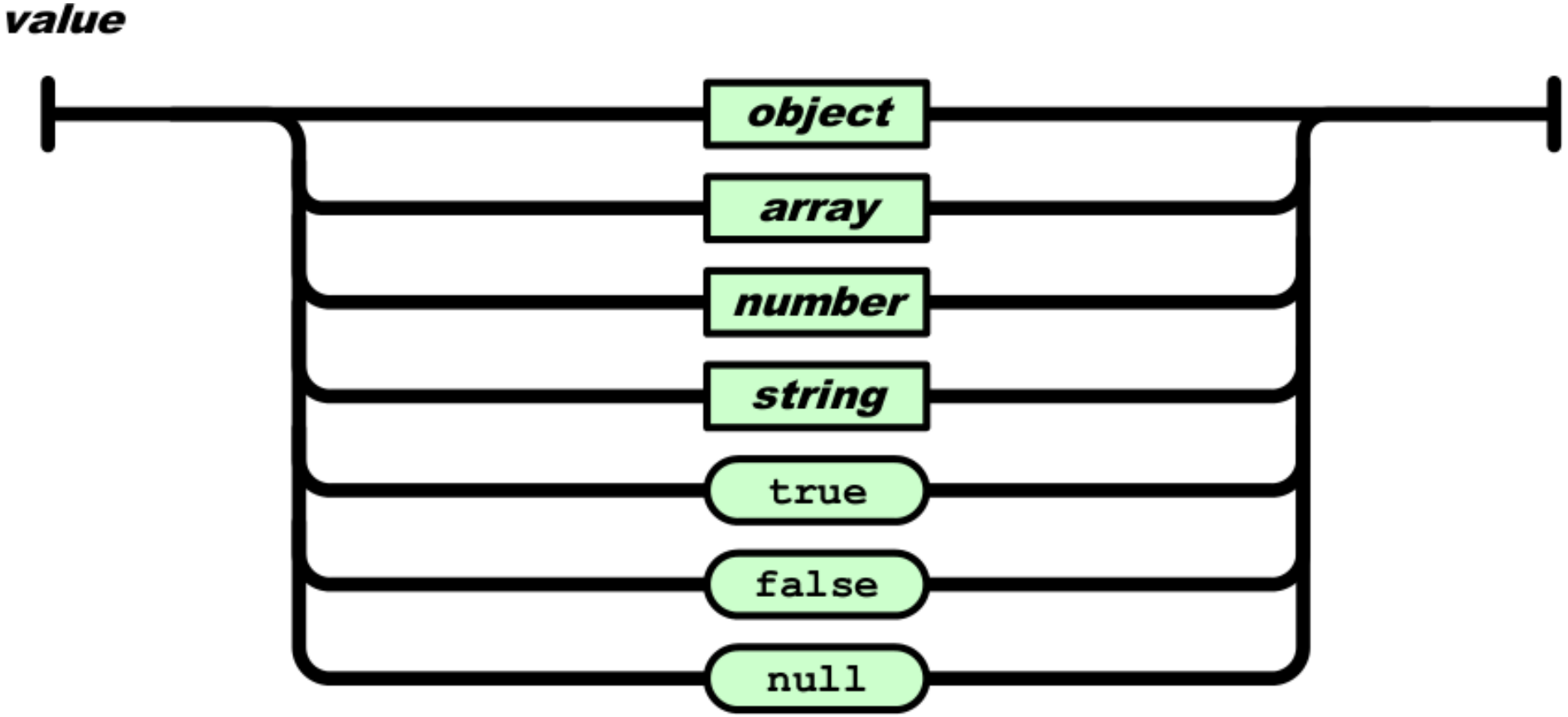
Data Types and Structures

- The following primitive types are available:
  - strings
  - numbers
  - booleans
  - `null`
- The following structured types are available:
  - arrays
  - objects

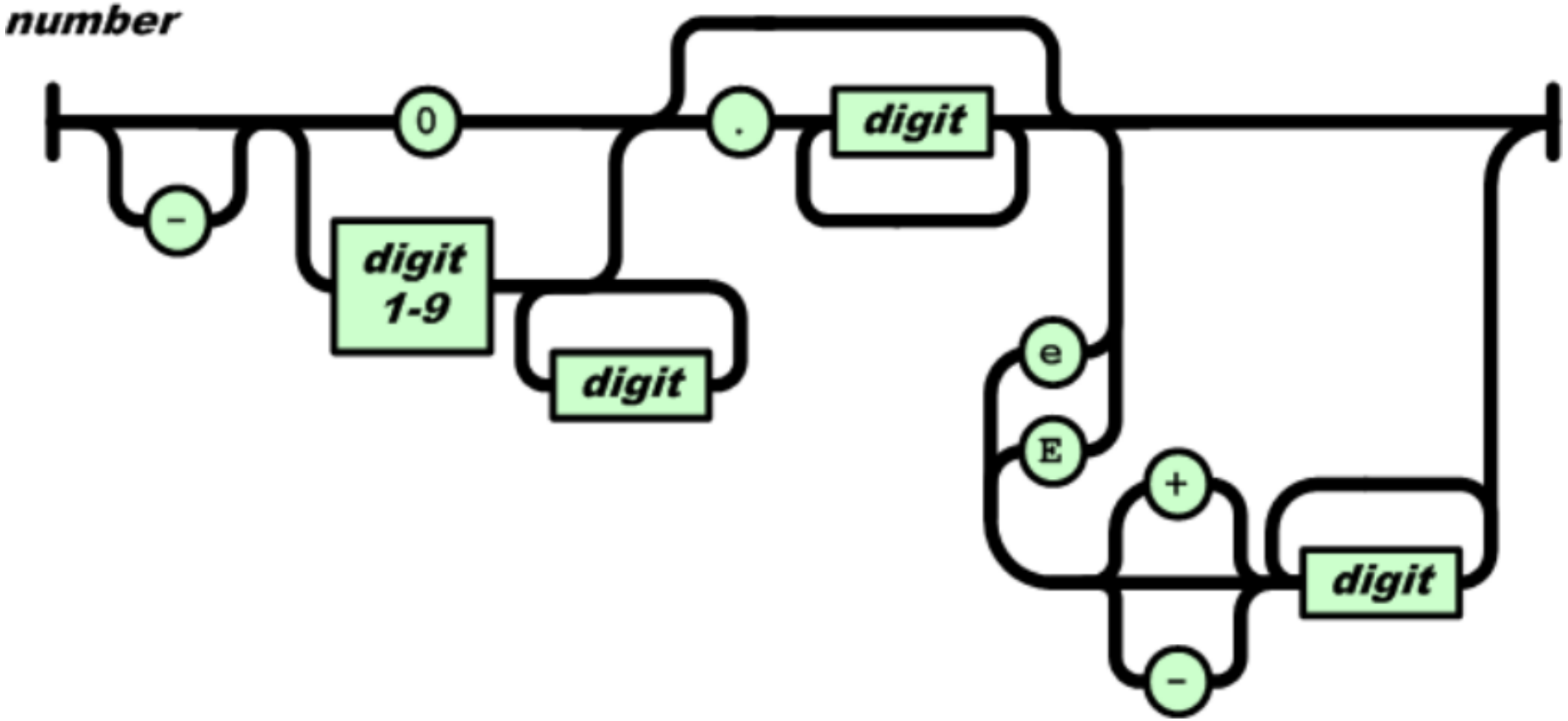
Tokens

- JSON text is a sequence of tokens the conforms to the JSON value grammar rule.
- Tokens:
  - The `{`, `}`, `[`, `]`, `:` and `,` characters are structural tokens.
  - Strings
  - Numbers
  - `true`, `false` and `null` are literal tokens.
- Whitespace characters are allowed before and after tokens and are insignificant.
  - Whitespace characters: HT (U+0009), LF (U+000A), CR (U+000D), space (U+0020).
  - Strings are the only tokens that can contain whitespace characters.

JSON Value



Numbers (1)



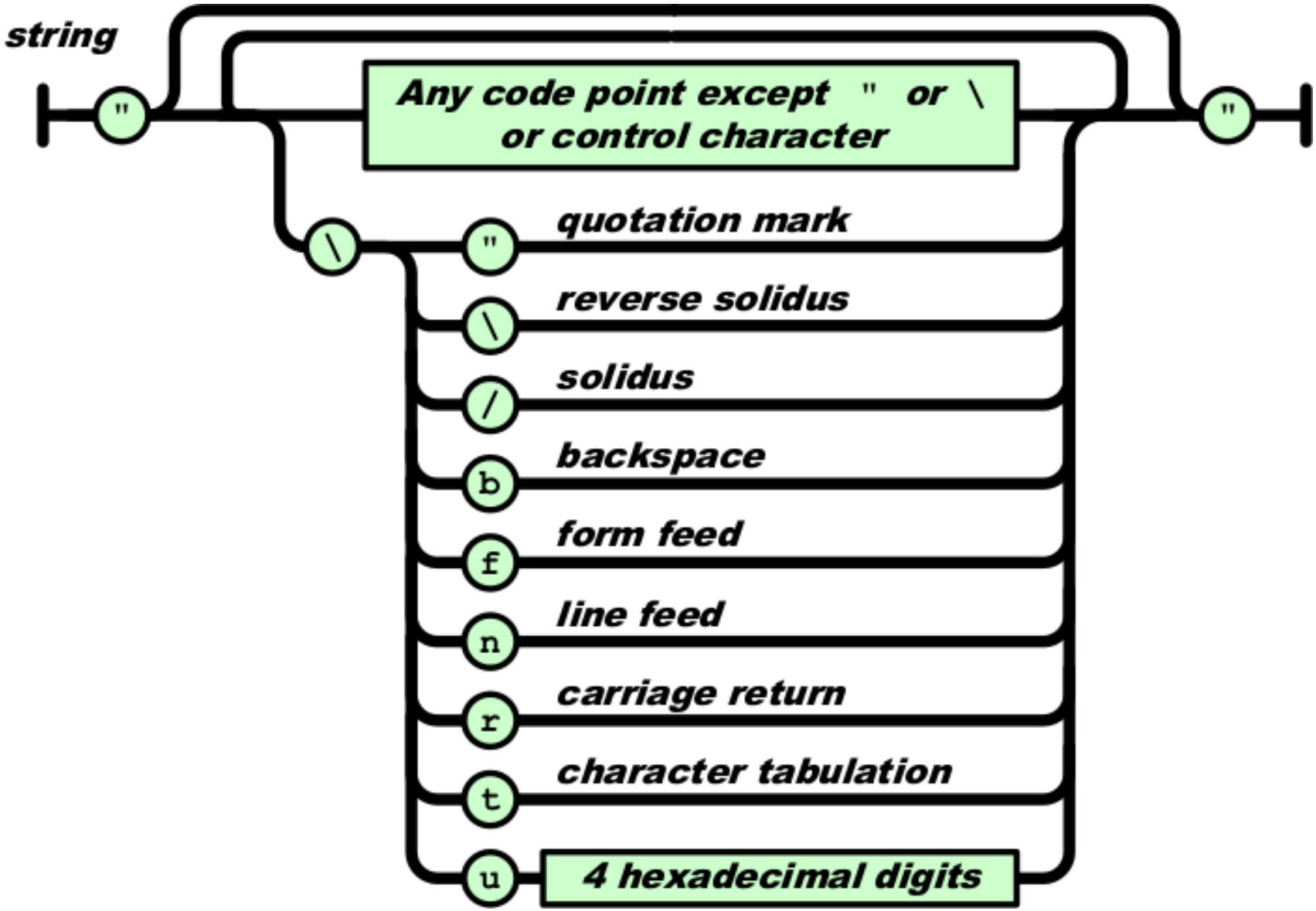
Numbers (2)

- No constraints are imposed on the range and precision of numbers.
- Implementations should consider interoperability.
  - For example, they should use double-precision numbers.
- Examples:
  - `0`
  - `-1.0`
  - `2.718282`
  - `1E-12`

Strings (1)

- Sequences of Unicode characters delimited by quotation marks (U+0022).
- Can contain any characters with the following exceptions that must be escaped:
  - Quotation mark (U+0022), backslash (U+005C), control characters (U+0000–U+001F)
- Special characters can be specified using escape sequences, such as `\"`, `\\`, `\t`, `\n`, `\r`.
- Unicode characters in the BMP can be specified as `\unnnn`, where `nnnn` is the code point of the character encoded by four hexadecimal digits.
- Examples: `""`, `"\"Hello, World!\n\""`, `"\u263A"`, `"\u263a"`

Strings (2)


string
"
Any code point except " or \
or control character
"
\
"
quotation mark
\
reverse solidus
/
solidus
b
backspace
f
form feed
n
line feed
r
carriage return
t
character tabulation
u
4 hexadecimal digits

Arrays (1)

- Ordered sequences of zero or more values (can be empty).
- Elements can be of any type (including arrays).

Arrays (2)

Examples:

- `["Athos", "Porthos", "Aramis", "d'Artagnan"]`
- `[9, 14, 19, 25, 26, 28]`
- `["Pi", 3.141593, null, true]`
- `[[45.7370889, 16.1133866], [48.5852340, 22.8981217]]`

Objects (1)

- Unordered collections of zero or more name/value pairs.
- A name is a string, a value is a JSON value.
- Name/value pairs are also called as members.

Objects (2)

RFC 8259:

- Objects with unique names are interoperable.
- Objects with non-unique names can be handled differently by applications.
- Implementations may or may not make the ordering of members available.

Objects (3)

Example:

```
{
  "title": "Alien",
  "year": 1979,
  "rating": 8.5,
  "votes": 981765,
  "genres": ["horror", "sci-fi"],
  "url": "https://www.imdb.com/title/tt0078748/"
}
```

Objects (4)

Example:

```
{
  "artist": "Porcupine Tree",
  "title": "Fear of a Blank Planet",
  "year": 2007,
  "tracks": [
    {
      "title": "Fear of a Blank Planet",
      "length": 448
    },
    {
      "title": "My Ashes",
      "length": 307
    },
    {
      "title": "Anesthetize",
      "length": 1062
    }
  ]
}
```

Objects (5)

Examples:

- Frankfurter https://www.frankfurter.app/
  - E.g., https://api.frankfurter.app/latest?base=HUF
- Chuck Norris Jokes API https://api.chucknorris.io/
  - E.g., https://api.chucknorris.io/jokes/random
- Nominatim https://nominatim.openstreetmap.org/ https://nominatim.org/
  - E.g., https://nominatim.openstreetmap.org/search?q=debrecen&format=json

Character Encoding

RFC 8259:

- JSON text exchanged between systems must be encoded using UTF-8.

Viewing JSON in Web Browsers

- Firefox: includes a built-in JSON viewer.
  - See: https://firefox-source-docs.mozilla.org/devtools-user/json_viewer/
- Chromium, Google Chrome:
  - Recommended extensions:
    - JSON Formatter https://chromewebstore.google.com/detail/json-formatter/bcjindcccaagfpapjjmafapmmgkkhgoa https://github.com/callumlocke/json-formatter
    - JSON Viewer Pro https://chromewebstore.google.com/detail/json-viewer-pro/eifflpmocdbdmepbjaopkkhbfmdgijcc https://github.com/rbrahul/Awesome-JSON-Viewer

Editors (1)

Free and open source software:

- JSON Editor (written in: JavaScript; license: Apache License 2.0) https://github.com/josdejong/jsoneditor/
  - Online: http://jsoneditoronline.org/
- Visual Studio Code (platform: Linux, macOS, Windows; license: MIT License) https://code.visualstudio.com/ https://github.com/Microsoft/vscode
  - See: *Editing JSON with Visual Studio Code* https://code.visualstudio.com/docs/languages/json

Editors (2)

Non-free software:

- <oXygen/> XML Editor (platform: Linux, macOS, Windows)
  https://www.oxygenxml.com/
  - See: https://www.oxygenxml.com/xml_editor/json_editor.html
- IntelliJ IDEA (platform: Linux, macOS, Windows)
  https://www.jetbrains.com/idea/
  - See: https://www.jetbrains.com/help/idea/json.html

Pretty-Printing JSON

- Pretty-printing JSON text on the command line:

```
python -m json.tool <file>
cat <file> | python -m json.tool
curl <url> | python -m json.tool
```

- Usage:

```
python -m json.tool --help
```

- See: https://docs.python.org/3/library/json.html#module-json.tool

Libraries

See: https://www.json.org/

- C++:
  - nlohmann/json (license: MIT License) https://json.nlohmann.me/ https://github.com/nlohmann/json
  - RapidJSON (license: MIT License) http://rapidjson.org/ https://github.com/Tencent/rapidjson
- Java:
  - Gson (license: Apache License 2.0) https://github.com/google/gson
  - Jackson (license: Apache License 2.0) https://github.com/FasterXML/jackson
- Python: the `json` module is part of the standard library https://docs.python.org/3/library/json.html

Submitting JSON in HTTP Requests: HTTPie (1)

Command line HTTP client.

- Website: https://httpie.io/
- Repository: https://github.com/httpie/cli
- Written in: Python
- Platform: Linux, macOS, Windows
- License: New BSD License

JSON support: https://httpie.io/docs/cli/json

Submitting JSON in HTTP Requests: HTTPie (2)

Example: creating a GitHub Gist on the command line

```
http https://api.github.com/gists \
  public:=true \
  description="Hello, World!" \
  files:='{"hello.txt": {"content": "Hello, World!"}}' \
  -a <username> \
  -v
```

See: https://docs.github.com/en/rest/gists

JSON Extensions

- JSON5 https://json5.org/ https://github.com/json5/json5
  - A proposed extension to JSON that aims to make it easier for humans to write and maintain by hand.
    - For example, adds new syntax features to JSON, such as comments; the names of members in objects can be strings or identifiers.
  - Is a superset of JSON, but does not introduce new data types.
  - Specification: *The JSON5 Data Interchange Format* https://spec.json5.org/
- YAML: YAML Ain't Markup Language https://yaml.org/
  - Is a superset of JSON.
  - Is more human readable than JSON, and offers a number of new features.
  - Specification: *YAML Ain't Markup Language (YAML) Version 1.2* https://yaml.org/spec/1.2.2/

BSON (“Binary JSON”)

- Binary data exchange format.
- Used by the MongoDB NoSQL database system: https://www.mongodb.org/
  - BSON is used for data storage and network transfer.
- Specification: https://bsonspec.org/
- Extends the types of JSON.
  - For example, timestamps, regular expressions.
  - There is no number type, uses the `int32`, `int64` and `double` types instead.

JSON Lines (1)

- A line-oriented format for storing one JSON value per line.
- Also called as newline-delimited JSON.
- Website: https://jsonlines.org/
- File extension: `.jsonl`
- See also: https://en.wikipedia.org/wiki/JSON_streaming

JSON Lines (2)

Examples:

- ```
  ["title", "year", "standalone"]
  ["Batman Begins", 2005, false]
  ["The Dark Knight", 2008, false]
  ["The Dark Knight Rises", 2012, false]
  ["Interstellar", 2014, true]
  ```

- ```
  { "title": "Batman Begins", "year": 2005, "standalone": false }
  { "title": "The Dark Knight", "year": 2008, "standalone": false }
  { "title": "The Dark Knight Rises", "year": 2012, "standalone": false }
  { "title": "Interstellar", "year": 2014, "standalone": true }
  ```

JSON Lines (3)

Uses: https://jsonlines.org/on_the_web/

JSON Lines (4)

Implementations:

- Java:
  - Gson (license: Apache License 2.0) https://github.com/google/gson
    - The class com.google.gson.JsonStreamParser can be used to read data in JSON Lines format.
  - Jackson jr (license: Apache License 2.0) https://github.com/FasterXML/jackson-jr
- Python:
  - jsonlines (license: New BSD License) https://jsonlines.readthedocs.io/ https://github.com/wbolster/jsonlines

Applications (1)

- Ajax (Asynchronous JavaScript and XML)
  - See: https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX

Applications (2)

Data interchange and storage:

- Firefox: exporting and importing bookmarks
  - See: *Restore bookmarks from backup or move them to another computer* https://support.mozilla.org/en-US/kb/restore-bookmarks-from-backup-or-move-them

Applications (3)

Data interchange and storage (continued):

- GeoJSON https://geojson.org/
  - A format for encoding geographic data structures.
  - Specification: Howard Butler, Martin Daly, Allan Doyle, Stefan Hagen, Tim Schaub. *RFC 7946: The GeoJSON Format*. August 2016. https://www.rfc-editor.org/rfc/rfc7946
- JSON-LD https://json-ld.org/
  - A lightweight syntax to serialize Linked Data in JSON.
  - Specification: *JSON-LD 1.1: A JSON-based Serialization for Linked Data* (W3C Recommendation, 16 July 2020) https://www.w3.org/TR/json-ld/

Applications (4)

Storing configuration data:

- `package.json`: http://package.json.is/ https://docs.npmjs.com/files/package.json
  - npm https://www.npmjs.com/ https://github.com/npm/cli
  - Grunt https://gruntjs.com/ https://github.com/gruntjs/grunt
    - See: https://gruntjs.com/getting-started#package.json
  - Visual Studio Code https://code.visualstudio.com/ https://github.com/microsoft/vscode
    - See: https://code.visualstudio.com/api/references/extension-manifest

Applications (5)

Storing configuration data (continued):

- Visual Studio Code (`settings.json`)
  - See: https://code.visualstudio.com/docs/getstarted/settings
- WebExtensions (`manifest.json`) https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions
  - See: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/manifest.json

Applications (6)

Web services:

- Meta Developer Documentation
  https://developers.facebook.com/docs/
- Flickr API https://www.flickr.com/services/api/
- GitHub REST API https://docs.github.com/en/rest/
- Nominatim API
  https://nominatim.org/release-docs/develop/api/Overview/

Applications (7)

NoSQL databases: a number of document-oriented databases use JSON for storing data.

- Apache CouchDB (written in: Erlang; platform: Linux, macOS, Windows; license: Apache License 2.0) https://couchdb.apache.org/ https://github.com/apache/couchdb
- EJDB2 (written in: C; platform: Android, iOS, Linux, macOS, Windows; license: MIT License) https://ejdb.org/ https://github.com/Softmotions/ejdb
- RethinkDB (written in: C++; platform: Linux, macOS; license: Apache License 2.0) https://rethinkdb.com/ https://github.com/rethinkdb/rethinkdb
- UnQLite (written in: C; platform: Linux, macOS, Windows; license: Simplified BSD License) https://unqlite.symisc.net/ https://github.com/symisc/unqlite

JSON Schema (1)

- A JSON-based schema language for validating JSON documents.
- Website: https://json-schema.org/
- Current version: `2020-12`
- The most widely supported version is still `draft-07` published in 2018.

JSON Schema (2)

Specifications:

- Austin Wright (ed.), Henry Andrews (ed.), Ben Hutton (ed.), Greg Dennis. *JSON Schema: A Media Type for Describing JSON Documents*. June 16, 2022. https://json-schema.org/draft/2020-12/json-schema-core
- Austin Wright (ed.), Henry Andrews (ed.), Ben Hutton (ed.). *JSON Schema Validation: A Vocabulary for Structural Validation of JSON*. June 16, 2022. http://json-schema.org/latest/json-schema-validation.html
- Geraint Luff, Henry Andrews (ed.), Ben Hutton (ed.). *Relative JSON Pointers*. January 28, 2020. https://datatracker.ietf.org/doc/html/draft-bhutton-relative-json-pointer-00

JSON Schema (3)

Related specification:

- Paul C. Bryan (ed.), Kris Zyp, Mark Nottingham (ed.), *RFC 6901: JavaScript Object Notation (JSON) Pointer*. April 2013. https://www.rfc-editor.org/rfc/rfc6901
  - Defines a syntax for identifying a specific value inside a JSON document.
  - Examples:
    - `/country`
    - `/places/0`
    - `/places/0/longitude`

JSON Schema (4)

- **JSON document**: an information resource described by the `application/json` media type, i.e., a JSON value.
- **Instance**: a JSON document to which a schema is applied.
- **JSON schema**: a JSON document used to describe an instance.
  - An object or a boolean.
  - Can be nested one into another.
    - The outermost schema is called the root schema, the others are called subschemas.
  - Media type: `application/schema+json`

JSON Schema (5)

- **Property**: a member of an object instance.
- **Keyword**: a property of a schema object that is applied to an instance.
  - Keywords assert constraints on JSON instances or annotate those instances with additional information.
  - Examples: `"properties"`, `"type"`, `"$ref"`

JSON Schema (6)

- **Vocabulary**: a set of keywords defined for a particular purpose together with their syntax and semantics.
  - Keywords have well defined syntax and semantics.
  - Each of the following specifications defines a vocabulary:
    - Austin Wright (ed.), Henry Andrews (ed.), Ben Hutton (ed.), Greg Dennis. *JSON Schema: A Media Type for Describing JSON Documents*. June 16, 2022. https://json-schema.org/draft/2020-12/json-schema-core
    - Austin Wright (ed.), Henry Andrews (ed.), Ben Hutton (ed.). *JSON Schema Validation: A Vocabulary for Structural Validation of JSON*. June 16, 2022. https://json-schema.org/draft/2020-12/json-schema-validation

JSON Schema (7)

- **Meta-schema**: a schema that itself describes a schema.
  - Example: JSON Schema meta-schema:
    - https://json-schema.org/draft/2020-12/schema
    - http://json-schema.org/draft-07/schema

JSON Schema (8)

Tools:

- Free and open source software:
  - Visual Studio Code (platform: Linux, macOS, Windows; license: MIT License) https://code.visualstudio.com/ https://github.com/Microsoft/vscode
    - See: *Editing JSON with Visual Studio Code* https://code.visualstudio.com/docs/languages/json
- Non-free software:
  - <oXygen/> XML Editor (platform: Linux, macOS, Windows) https://www.oxygenxml.com/
    - See: *Editing JSON Schema Documents* https://www.oxygenxml.com/doc/versions/26.1/ug-editor/topics/editing-JSON-schema.html
  - IntelliJ IDEA (platform: Linux, macOS, Windows) https://www.jetbrains.com/idea/
    - See: https://www.jetbrains.com/help/idea/json.html

JSON Schema (9)

Implementations: https://json-schema.org/tools

- C++:
  - json-schema-validator (license: MIT License)
    https://github.com/pboettch/json-schema-validator
- Java:
  - json-schema (license: Apache License 2.0)
    https://github.com/everit-org/json-schema
- JavaScript:
  - Ajv (license: MIT License) https://ajv.js.org/
    https://github.com/ajv-validator/ajv

JSON Schema (10)

Implementations: https://json-schema.org/tools

- .NET:
  - Json.NET Schema (license: AGPLv3)
    http://www.newtonsoft.com/jsonschema
    https://github.com/JamesNK/Newtonsoft.Json.Schema
    - Web interface: https://www.jsonschemavalidator.net/
- Python:
  - jschon (license: MIT License) https://jschon.readthedocs.io/en/latest/
    https://github.com/marksparkza/jschon
    - Web interface: https://jschon.dev/
  - jsonschema (license: MIT License)
    https://python-jsonschema.readthedocs.io/
    https://github.com/python-jsonschema/jsonschema

JSON Schema (11)

JSON Schema Store https://www.schemastore.org/json/
https://github.com/SchemaStore/schemastore

- A collection of schemas for commonly known JSON file formats.
- Editor support: IntelliJ IDEA, Microsoft Visual Studio, ...

JSON Schema (12)

Associating JSON schemas with JSON documents:

- JSON schema specifications do not provide any document level means.
- Implementation specific solutions:
  - <oXygen/> XML Editor: https://www.oxygenxml.com/doc/versions/26.1/ug-editor/topics/json-associating-schema-directly-in-doc.html
  - Visual Studio Code: https://code.visualstudio.com/docs/languages/json#_json-schemas-and-settings

JSON Schema (13)

Further recommended reading:

- Michael Droettboom, *Understanding JSON Schema*.
  https://json-schema.org/understanding-json-schema/
  https://github.com/json-schema-org/understanding-json-schema

JSON Schema Examples (1)

- Schema:
  - `true`
  - `{}`
- Valid instances: any JSON value
- Invalid instances: none

- Schema:
  - `false`
  - `{ "not": {} }`
- Valid instances: none
- Invalid instances: any JSON value

JSON Schema Examples (2)

Schema:

```
{
  "type": "string"
}
```

Valid instances:

- `""`
- `"Hello, World!\n"`

JSON Schema Examples (3)

Schema:

```
{
  "type": ["string", "null"]
}
```

Valid instances:

- `""`
- `"Hello, World!\n"`
- `null`

JSON Schema Examples (4)

Schema:

```
{
  "type": "array",
  "items": { "type": "string" }
}
```

Valid instances:

- `[]`
- `["Hello, World!\n"]`
- `["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"]`

JSON Schema Examples (5)

Schema:

```
{
  "type": "array",
  "items": { "type": "string" },
  "minItems": 1,
  "uniqueItems": true
}
```

Valid instances:

- `["sci-fi"]`
- `["crime", "drama"]`

Invalid instances:

- `[]`
- `["sci-fi", "sci-fi"]`

JSON Schema Examples (6)

Schema:

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "name": { "type": "string" },
    "age": { "type": "integer" },
    "email": { "type": "string" },
    "webpage": { "type": "string" }
  },
  "required": ["name", "age", "email"]
}
```

JSON Schema Examples (7)

A valid instance (continued):

```
{
  "name": "Douglas Crockford",
  "age": 69,
  "email": "douglas@crockford.com",
  "webpage": "https://www.crockford.com/"
}
```

JSON Schema Examples (8)

Schema (continued):

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "name": { "type": "string" },
    "age": { "type": "integer", "minimum": 0 },
    "email": { "type": "string", "format": "email" },
    "webpage": { "type": "string", "format": "uri",
      "pattern": "^http(s)?://.*"
    }
  },
  "required": ["name", "age", "email"]
}
```

JSON Schema Examples (9)

Schema:

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "title": { "type": "string" },
    "year": { "type": "integer" },
    "rating": { "type": "number", "minimum": 0, "maximum": 10 },
    "votes": { "type": "integer", "minimum": 0 },
    "genres": {
      "type": "array",
      "items": { "type": "string" },
      "minItems": 1,
      "uniqueItems": true
    },
    "url": { "type": "string", "format": "uri" }
  },
  "required": ["title", "year", "rating", "votes", "genres", "url"],
  "additionalProperties": false
}
```

JSON Schema Examples (10)

Schema:

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "artist": { "type": "string" },
    "title": { "type": "string" },
    "year": { "type": "integer" },
    "tracks": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "title": { "type": "string" },
          "length": { "type": "integer", "minimum": 0 }
        },
        "required": ["title", "length"],
        "additionalProperties": false
      },
      "minItems": 1
    }
  },
  "required": ["artist", "title", "year", "tracks"],
  "additionalProperties": false
}
```

JSON Schema Examples (11)

Schema:

```
{
  "$schema": "http://json-schema.org/2020-12/schema",
  "type": "object",
  "patternProperties": {
    "^[a-z]{2}$": { "type": "string" }
  },
  "additionalProperties": false,
  "minProperties": 2
}
```

A valid instance:

```
{
  "en": "Lord of the Rings",
  "de": "Der Herr der Ringe",
  "hu": "A gyűrűk ura"
}
```

JSON Schema Examples (12)

Schema:

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "name": { "type": "string" },
    "born": { "$ref": "#/$defs/event" },
    "died": { "$ref": "#/$defs/event" },
    "gender": { "enum": ["female", "male"] }
  },
  "required": ["name", "born", "gender"],
  "$defs": {
    "event": {
      "type": "object",
      "properties": {
        "date": { "type": "string" },
        "place": { "type": "string" }
      }
    }
  }
}
```

JSON Schema Examples (13)

A valid instance (continued):

```
{
  "name": "Edgar Allan Poe",
  "born": {
    "date": "1809-01-19",
    "place": "Boston, Massachusetts, United States"
  },
  "died": {
    "date": "1849-10-07",
    "place": "Baltimore, Maryland, United States"
  },
  "gender": "male"
}
```

JSON Schema Examples (14)

Schema:

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "name": { "type": "string" },
    "born": { "$ref": "#/$defs/event" },
    "died": { "$ref": "#/$defs/event" },
    "gender": { "enum": ["female", "male"] }
  },
  "required": ["name", "born", "gender"],
  "$defs": {
    "event": {
      "type": "object",
      "properties": {
        "date": { "type": "string" },
        "place": { "type": "string" }
      },
      "anyOf": [
        { "required": ["date"] },
        { "required": ["place"] }
      ]
    }
  }
}
```

JSON Schema Examples (15)

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "name": { "type": "string" },
    "born": { "$ref": "#/$defs/event" },
    "died": {
      "allOf": [
        { "$ref": "#/$defs/event" },
        {
          "properties": {
            "cause": { "type": "string" }
          },
          "required": ["cause"]
        }
      ]
    },
    "gender": { "enum": ["female", "male"] }
  },
  "required": ["name", "born", "gender"],
  "$defs": {
    "event": {
      "type": "object",
      "properties": {
        "date": { "type": "string" },
        "place": { "type": "string" }
      }
    }
  }
}
```

JSON Schema Examples (16)

A valid instance (continued):

```
{
  "name": "John F. Kennedy",
  "born": {
    "date": "1917-05-29",
    "place": "Brookline, Massachusetts, United States"
  },
  "died": {
    "date": "1963-11-22",
    "place": "Dallas, Texas, United States",
    "cause": "assassination"
  },
  "gender": "male"
}
```

JSON Schema Examples (17)

Schema:

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "type": { "enum": ["book", "serial"] },
    "title": { "type": "string" },
    "publisher": { "type": "string" }
  },
  "required": ["type", "title", "publisher"],
  "if": {
    "properties": { "type": { "const": "book"} }
  },
  "then": {
    "properties": {
      "isbn": { "type": "string", "pattern": "^[0-9]{13}$" }
    },
    "required": ["isbn"]
  },
  "else": {
    "properties": {
      "issn": { "type": "string", "pattern": "^[0-9]{4}-[0-9]{3}[0-9X]$" }
    },
    "required": ["issn"]
  }
}
```

JSON Schema Examples (18)

Valid instances:

- ```
  {
    "type": "book",
    "title": "The Hound of the Baskervilles",
    "publisher": "Penguin Books",
    "isbn": "9780241952870"
  }
  ```
- ```
  {
    "type": "serial",
    "title": "IEEE Internet of Things Journal",
    "publisher": "IEEE",
    "issn": "2327-4662"
  }
  ```

JSON Schema Examples (19)

Schema: tuple validation

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "array",
  "prefixItems": [
    {
      "type": "number",
      "minimum": -90,
      "maximum": 90
    },
    {
      "type": "number",
      "minimum": -180,
      "maximum": 180
    }
  ],
  "items": false
}
```

JSON Schema Examples (20)

Real examples:

- GeoJSON: https://json.schemastore.org/geojson
- `manifest.json` (WebExtensions): https://json.schemastore.org/webextension
- `manifest.json` (Chromium, Google Chrome): https://json.schemastore.org/chrome-manifest
- `package.json` (npm): https://json.schemastore.org/package
- SWAPI – The Star Wars API https://swapi.dev/
  - https://swapi.dev/api/planets/schema
  - https://swapi.dev/api/species/schema
  - https://swapi.dev/api/starships/schema

JSON Schema Use Cases

Web UI generation from JSON schema:

- Alpaca (written in: JavaScript; license: Apache License 2.0) http://www.alpacajs.org/ https://github.com/gitana/alpaca
- jsonforms (written in: TypeScript; license: MIT License) https://jsonforms.io/ https://github.com/eclipsesource/jsonforms
- react-jsonschema-form (written in: JavaScript; license: Apache License 2.0) https://rjsf-team.github.io/react-jsonschema-form/ https://github.com/rjsf-team/react-jsonschema-form

See: https://json-schema.org/tools?query=&sortBy=name&sortOrder=ascending&groupBy=toolingTypes&licenses=&languages=&drafts=&toolingTypes=schema-to-web-UI

XML-JSON Conversion (1)

- <oXygen/> XML Editor https://www.oxygenxml.com/
  - Tools > JSON to XML...
  - Tools > XML to JSON...
- Visual Studio Code
  - Plugin: XML to JSON (license: MIT License) https://marketplace.visualstudio.com/items?itemName=buianhthang.xml2json https://github.com/anhthang/vscode-xml2json

XML-JSON Conversion (2)

- JSON-java (written in: Java; license: JSON License)
  https://github.com/stleary/JSON-java
  - The `JSONObject()` method of the `XML` class converts an XML document to an equivalent JSON object.
- xml-js (written in: JavaScript; license: MIT License)
  https://github.com/nashwaan/xml-js

XML-JSON Conversion (3)

- JsonML (JSON Markup Language) http://www.jsonml.org/ https://github.com/mckamey/jsonml
  - Aims the lossless conversion of XML documents to JSON.
  - Syntax: http://www.jsonml.org/syntax/

Query Languages (1)

- XPath, XQuery: JSON support was added in the latest version (version 3.1).
  - Further information: https://www.w3.org/XML/Query/

Query Languages (2)

There exist a number of custom languages, I personally find the following of them promising:

- JSONiq https://www.jsoniq.org/
  - Declarative functional language based on XQuery for querying and processing JSON.
  - Implementations:
    - RumbleDB (written in: Java; license: Apache License 2.0) https://rumbledb.org/ https://github.com/RumbleDB/rumble
- JSONata (license: MIT License) https://jsonata.org/ https://github.com/jsonata-js/jsonata
  - Query and transformation language inspired by the semantics of XPath 3.1 path expressions.
  - A reference implementation in JavaScript.

Query Languages (3)

There exist a number of custom languages, I personally find the following of them promising: (continued)

- jq (written in: C; platform: Linux, macOS, Windows; license: Expat License) https://jqlang.github.io/jq/ https://github.com/jqlang/jq
  - Command line JSON processor.
- JMESPath https://jmespath.org/ https://github.com/jmespath
  - Implementations: Go, Java, JavaScript, Lua, .NET, PHP, Python, Ruby, Rust (license: MIT License)
- ObjectPath (written in: Python; license: MIT License) http://objectpath.org/ https://github.com/adriank/ObjectPath

Acknowledgments

Thanks for László Szathmáry for his remarks and for recommending jq.