JSON Schema Linting Rules

Table of Contents

The initial rules solve immediate, common pitfalls and provide foundational improvements to JSON Schema quality based on my findings in Stack Overflow, https://github.com/json-schema-org/json-schema-spec/issues, Slack and other discussions. Several rules align closely with official JSON Schema recommendations (learnjsonschema.com). I have also stated additional 5 rules that may be addressed.

The rules are stated in the format of https://github.com/orgs/json-schema-org/discussions/323#discussioncomment-4903922

- Rule name
- description
- examples of invalid
- examples of valid
- how to fix
- default severity
- reference
- why picked

---

1. Enforce `$schema` Keyword at the Root

- **Description**: Ensure every schema explicitly specifies the JSON Schema draft version through the `$schema` keyword at the root.
- **Examples of Invalid**:

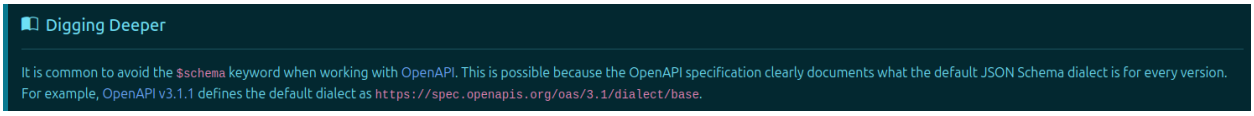
```
{
  "type": "object",
  "properties": {}
}
```

- **Examples of Valid**:

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {}
}
```

- **How to Fix**: Add the `$schema` keyword explicitly indicating the schema draft version.
- **Default Severity**: **Warning** (The absence doesn't break validation directly but introduces ambiguity regarding schema interpretation and compatibility with different validators.).
- **Reference**: https://json-schema.org/understanding-json-schema/reference/schema
- **Why Picked**:
  Explicit
  `$schema` declarations improve clarity and ensure validators interpret schemas correctly, aligning with the JSON Schema official recommendations, eliminating ambiguity about schema interpretation.

Note: In contexts where schemas are embedded within larger specifications (e.g., OpenAPI), requiring `$schema` at the root may not be feasible or necessary. Linting tools should allow this rule to be configurable based on schema usage context. It is mentioned in https://www.learnjsonschema.com/2020-12/core/schema/



---

2. Avoid Combining `enum` with Other Validation Keywords

- **Description**: Prevent usage of `enum` in combination with other validation keywords like `type`, `minimum`, or `maxLength`.
- **Examples of Invalid**:

```
{
  "type": "string",
  "enum": ["one", "two"],
  "minLength": 3
}
```

- **Examples of Valid**:

```
{
  "enum": ["one", "two"]
}
```

- **How to Fix**:

  Instead of removing conflicting validation rules outright, each keyword conflict would be evaluated individually. For this example:

  - First, the validity of the `enum` itself would be checked to ensure that the values it contains are valid.
  - Subsequently, the conflicting validation rule (like `minLength`) would be validated against the type of values present in the `enum`. If `minLength` is a constraint that applies to a string type, we would check if any values in the `enum` fail this validation. In case of failure, the conflicting keyword (e.g., `minLength`) would be flagged for removal.
  - The remaining valid rules would be kept, ensuring the schema still adheres to best practices while maintaining its integrity.

- **Default Severity**: **Error** (Combining these keywords results in schema conflicts, causing unexpected validation behaviors or failures.).
- **Reference**: https://github.com/orgs/json-schema-org/discussions/323#discussioncomment-4898765
- **Why Picked**:
  Combining
  `enum` with additional constraints leads to redundant and potentially conflicting validations, complicating schema logic and validation. `enum` itself restricts

allowable values, making additional constraints unnecessary and potentially contradictory.

---

3. Use `const` Instead of Single-Value `enum`

- **Description**: Use `const` rather than single-value `enum` for improved semantic clarity.
- **Examples of Invalid**:

```
{
 "enum": ["foo"]
}
```

- **Examples of Valid**:

```
{
 "const": "foo"
}
```

- **How to Fix**: Replace single-value `enum` declarations with `const`.
- **Default Severity**: **Warning** or **Info**(Best practice recommendation, primarily improves readability and schema clarity; does not cause functional validation issues.).
- **Reference**: General best practice (from JSON Schema linting discussions and learnjsonschema.com), https://github.com/sourcemeta/jsonschema/pull/228#discussion_r1973520202
- **Why Picked**:
  Single-value enums can obscure the schema's intent. The `const` keyword clearly communicates that only one value is acceptable.

---

4. Disallow Use of Invalid or Non-standard Keywords

- **Description**: Ensure that all keywords used in the schema are valid for the declared `$schema` version (dialect). JSON Schema does not formally deprecate

keywords — a keyword either exists in a given dialect or it doesn't. Use of unknown or unsupported keywords should be flagged, and optionally, prefixed with `x-` to denote custom extensions.

The only exception is `definitions`, which is *not invalid* but discouraged in favor of `$defs` starting from draft-2019-09.

- **Examples of Invalid**:

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "credit_card": { "type": "string" }
  },
  "dependencies": {
    "credit_card": ["billing_address"]
  },
  "customValidation": true
}
```

- **Examples of Valid**:

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "credit_card": { "type": "string" }
  },
  "dependentRequired": {
    "credit_card": ["billing_address"]
  },
  "x-customValidation": true
}
```

- **How to Fix**: Remove or prefix it with `x-` (which will be the new mandatory prefix for unknown keywords in the next version of JSON Schema)

- **Default Severity**: **Error** (compatibility issue) and **Warning** – Use of discouraged keywords like `definitions` in newer drafts.
- **Reference**: JSON Schema Spec Discussion #1079, The JSON Schema 2019-09 release notes indicate that the `dependencies` keyword has been split into `dependentSchemas` and `dependentRequired`, JSON Schema draft consistency guidelines.
- **Why Picked**:
  Avoiding deprecated keywords ensures schemas remain valid across JSON Schema specification updates.

Note: The only exception is `definitions`, which is *not invalid* but discouraged in favor of `$defs` starting from draft-2019-09.

---

5. Validate Type-Specific Keyword Usage

- **Description**: Ensure validation keywords (`minimum`, `maxLength`, `pattern`) align with compatible data types.
- **Examples of Invalid**:

```
{
  "type": "string",
  "minimum": 5
}
```

- **Examples of Valid**:

```
{
  "type": "string",
  "maxLength": 5
}
```

- **How to Fix**: Delete, as guessing could lead to unintended surprises
- **Default Severity**: **Error** (schema validation failure).

- **Reference**: Stack Overflow Issue . https://json-schema.org/understanding-json-schema/reference/type
- **Why Picked**:
  Prevents invalid schema definitions and runtime validation errors. Ensuring keywords match their correct data types prevents schema errors. Keywords like
  `minimum` and `maximum` should be applied only to numeric data types, `maxLength` only to strings, etc.

---

6. Require Explicit Definitions for Arrays

- **Description**: Mandate explicit definitions (`items`) for schemas defining arrays to specify expected data structures.
- **Examples of Invalid**:

```
{
  "type": "array"
}
```

- **Examples of Valid**:

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

- **How to Fix**: Add explicit `items` definitions.
- **Default Severity**: **Warning** (clarity and validation recommendation as not explicitly defining array contents causes ambiguity but doesn't directly break schema validation.).
- **Reference**: JSON Schema official guidance on arrays.
- **Why Picked**:
  Clearly defines expected array contents, avoiding ambiguous or unintended

data.

---

7. Explicitly Define `additionalProperties`

- **Description**: Control allowance of properties not explicitly defined in the schema.
- **Examples of Invalid**:

```
{
  "type": "object",
  "properties": {
    "name": {"type": "string"}
  }
}
```

- **Examples of Valid**:

```
{
  "type": "object",
  "properties": {
    "name": {"type": "string"}
  },
  "additionalProperties": false
}
```

- **How to Fix**: Explicitly set `additionalProperties` to `false` if additional properties aren't desired.
- **Default Severity**: **Warning** (data integrity best practice).
- **Reference**: Stack Overflow Q&A
- **Why Picked**:
  Prevents unintended additional data, ensuring data integrity.

---

8. Avoid Deeply Nested Schemas

- **Description**: Limit the depth of schema nesting to maintain clarity and readability.
- **Examples of Invalid**: Schemas with more than three nested objects.
- **Examples of Valid**: Schemas limited to a manageable nesting level (max 3).
- **How to Fix**: Refactor deeply nested schemas into separate reusable definitions.
- **Default Severity**: **Warning** (maintainability).
- **Reference**: Stack Overflow Discussion, https://json-schema.org/understanding-json-schema/structuring
- **Why Picked**:
  Improves readability and simplifies schema maintenance.

---

9. Ensure Schema Elements are Documented

- **Description**: Encourage documenting schema elements using `description` and `title`.
- **Examples of Invalid**: Missing documentation fields.
- **Examples of Valid**:

```
{
  "type": "object",
  "properties": {
    "email": {
      "type": "string",
      "description": "Primary user email."
    }
  }
}
```

- **How to Fix**: Add descriptive annotations to schema elements.
- **Default Severity**: **Info** (documentation recommendation).
- **Reference**: JSON Schema Documentation Guidelines.

- **Why Picked**:
  Enhances schema understanding for developers and maintainers.

Additional Potential Rules

10. Enforce Consistency of `required` Keyword

- **Description**: Prevent schemas from declaring required properties that aren't defined, avoiding schema validation errors.
- **Reference**: https://json-schema.org/draft/2019-09/draft-handrews-json-schema-validation-02

11. Restrict Overuse of `anyOf`, `oneOf`, and `allOf`

- **Description**: Prevent overly complex schema logic, making validation simpler and schema easier to understand.
- **Reference**: https://json-schema.org/understanding-json-schema/reference/combining

12. Specify Appropriate `format` for String Types

- **Description**: Enhances data validation by enforcing known string formats (email, URL, etc.), reducing the burden on application logic.
- **Reference**: String Formats in JSON Schema

13. Avoid Overly Permissive Schemas

- **Description**: Prevent unintended acceptance of any data, ensuring schema effectively constrains allowed data.
- **Reference**: https://stackoverflow.com/questions/17530762/only-allow-properties-that-are-declared-in-json-schema

14. Restrict Usage of Broad Types

- **Description**: Encourages schema precision, improving validation accuracy and readability.

- **Reference**: https://json-schema.org/understanding-json-schema/reference/combining

---

Identification and Auto-fix Summary for some of the Rules:

1. **Enforce `$schema` Keyword at the Root**:
   - **Identification**:
     - *Manual*: Check if the root object includes the `$schema` keyword.
     - *Automated*: Linting tools can flag schemas missing the `$schema` keyword. (JSON Schema CLI's `lint` command to check for the presence of the `$schema` keyword at the root level of schemas. The linter can flag schemas missing this declaration.)
   - **Auto-fix**:
     - When the `$schema` keyword is absent, the linter can automatically insert a default `$schema` declaration, such as `"$schema": "https://json-schema.org/draft/2020-12/schema"`, to specify the schema version.
2. **Avoid Combining `enum` with Other Validation Keywords**:
   - **Identification**:
     - *Manual*: Look for `enum` used alongside other constraints like `type`, `minimum`, etc.
     - *Automated*: The linter can analyze schemas to detect instances where the `enum` keyword is combined with other validation keywords like `type`, `minimum`, or `maxLength`, which may result in conflicting validations.
   - **Auto-fix**:
     - The linter can suggest or automatically remove conflicting validation keywords when used alongside `enum`, ensuring that the schema's intent is clear and unambiguous.
3. **Use `const` Instead of Single-Value `enum`**:
   - **Identification**:

    - *Manual*: Identify `enum` arrays containing only one value.
    - *Automated*: The linter can identify `enum` arrays containing a single value, where the use of the `const` keyword would be more appropriate to express the schema's intent.
- **Auto-fix**:
    - Replace `{"enum": ["value"]}` with `{"const": "value"}`.
