JSON Schema

# JSON

JavaScript Object Notation (JSON) is a simple data interchange format. Orignally used for the world wide web, since JS is used throughout web, but its now used throughout many different computer systems.

## JSON Components

* object - { "key1": "value1", "key2": "value2" }
* array - [ "first", "second", "third" ]
* number - 42, 3.141
* string - "a string"
* boolean - true
* null - null

# Schema

A JSON schema decribes how a JSON object should be structured. Since the schema is also written in JSON format, it can be used in software to automatically validate structure. However, this does come with the weakness of not being able to contain abirary code which could be used to validate against certain constraints or relationships between data elements which can't be expressed in JSON format.

## Basic schemas

The most basic schema is an empty object, which will accept any valid JSON object. The boolean true can also be used to validate any JSON, false can be used to do the opposite.

### $schema

The "$schema" keyword is used to declare that a JSON is a JSON Schema, and which version of the JSON Schema specification it is written to. For example, a basic object written in the most recent:

{ "$schema": "http://json-schema.org/draft-07/schema#'}

### $id

The "$id" keword is used to set a unique identifier for each schema. The identifier is a URI-reference, so it is common to set it to a URL at a domain you control. This provides a unique id and normally indicates where the schema can be downloaded so others can validate their objects against your schema. For example:

{ "$id": "http://foo.bar/schemas/address.json" }

The $id identifier also declares a base URI against which $ref URI-references are resolved. Therefore, if a "$ref" reference is made to another schema, such as:

{ "$ref": "person.json" }

A validation library which supports network fetching may fetch the person.json schema from:

http://foo.bar/schemas/person.json

$id can also be used in a schema sublevel to allow references to the subschema using id, without using JSON pointers. For example:

{

"$schema": "http://json-schema.org/draft-07/schema#",

"definitions": {

"address": {

"$id": "#address",

"type": "object",

"properties": {

"street\_address": { "type": "string" },

"city": { "type": "string" },

"state": { "type": "string" }

},

"required": ["street\_address", "city", "state"]

}

},

"type": "object",

"properties": {

"billing\_address": { "$ref": "#address" },

"shipping\_address": { "$ref": "#address" }

}

}

$ref can also be used with the combining keywords to extend schema:

* allOf - must be valid against all of the subschemas
* anyOf - must be valid against any of the subschemas
* oneOf - must be valid against exactly one of the subschemas

For example:

"allOf": [

{ "$ref": "#/definitions/address" },

{ "properties": {

"type": { "enum": [ "residential", "business" ] }

}

}

]

## Reference

### Type

The 'type' keyword is used to restrict a value to a specific type. For example, to restrict a basic JSON to the 'string' type:

{ "type": "string" }

Will validate:

"a string"

and invalidate:

42

Types are any JSON components:

* object - { "key1": "value1", "key2": "value2" }
  + properties - defines properties of object eg:
    - "street\_name": { "type": "string" },
  + additionalProperties - by default any additional properties are allowed, set to false to prevent any, set to object to define what they should be, e.g.:
    - "additionalProperties": { "type": "string" }
  + required - lists properties out of properties object which are required
  + propertyNames - pattern to match properties names:
    - "pattern": "^[A-Za-z\_][A-Za-z0-9\_]\*$"
  + dependencies - schema changes based on the presence of certain special properties
  + patternProperties - given specific name, apply specific schema
* array - [ "first", "second", "third" ] by default items can be anything
  + items - schema for array items to match, can be single or array, if array it will match in order, then allow any item after.
  + contains - schema to validate against one or more items in array
  + additionalItems - false to prevent any additional from 'items', schema to validate any additional against
  + minItems
  + maxItems
  + uniqueItems
* number/integer - 42, 3.141 (number is any numeric type)
  + multipleOf
  + minimum - x >=
  + exclusiveMinimum - x >
  + maximum - x <=
  + exclusiveMaximum - x <
* string - "a string"
  + minLength
  + maxLength
  + pattern - match a regex <https://json-schema.org/understanding-json-schema/reference/regular_expressions.html>
  + format
    - date-time
    - email
    - hostname
    - ipv4
    - uri
  + <https://json-schema.org/understanding-json-schema/reference/string.html>
* boolean - true
* null - null

### Annotations

Annotations are keywords which are not strictly used for validation, but can be used to describe parts of the schema.

* title
* description
* default
* examples - array of examples

### Comments

$comments can be used to add comments which must be a string.

### enum

The "enum" keyword is used to restrict a value to a fixed set of values, it must be an array with atleast one element and all others being unique. For example:

"enum": ["red", "amber", "green", null, 42]

will match, 'red' but not 'blue'

### Constants

The 'const' keyword is used to restrict a value to a single value. For example:

"country": {

"const": "United States of America"

}

Will only match a country value of 'United States of America'

### Media

Media keywords can be used to validate content type if the validation library supports it.

* contentMediaType - specifies the MIME type of the contents of the string
* contentEncoding - type of encoding used to store the contents:
  + 7bit
  + 8bit
  + binary
  + quoted-printable
  + base64

For example, an encoded image:

{

"type": "string",

"contentEncoding": "base64",

"contentMediaType": "image/png"

}

### Conditionals

The if, then, and else keywords allow the application of a subschema based on the outcome of another schema. For example:

{

"type": "object",

"properties": {

"street\_address": {

"type": "string"

},

"country": {

"enum": ["United States of America", "Canada"]

}

},

"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]" } }

}

}

Multiple conditionals can be wrapped in allOf statements to allow specific values to scale.