Internet Engineering Task Force	K. Zyp, Ed.	
Internet-Draft	SitePen (USA)	
Intended status: Informational	G. Court	
Expires: May 26, 2011	November 22, 2010	

TOC

A JSON Media Type for Describing the Structure and Meaning of JSON Documents
draft-zyp-json-schema-03

Abstract

JSON (JavaScript Object Notation) Schema defines the media type "application/schema+json", a JSON based format for defining the structure of JSON data. JSON Schema provides a contract for what JSON data is required for a given application and how to interact with it. JSON Schema is intended to define validation, documentation, hyperlink navigation, and interaction control of JSON data.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction
2. Conventions
3. Overview
    3.1. Terminology
    3.2. Design Considerations
4. Schema/Instance Association
    4.1. Self-Descriptive Schema
5. Core Schema Definition
    5.1. type
    5.2. properties
    5.3. patternProperties
    5.4. additionalProperties
    5.5. items
    5.6. additionalItems
    5.7. required
    5.8. dependencies
    5.9. minimum
    5.10. maximum
    5.11. exclusiveMinimum
    5.12. exclusiveMaximum
    5.13. minItems
    5.14. maxItems
    5.15. uniqueItems
    5.16. pattern
    5.17. minLength
    5.18. maxLength
    5.19. enum
    5.20. default
    5.21. title
    5.22. description
    5.23. format
    5.24. divisibleBy
    5.25. disallow
    5.26. extends
    5.27. id
    5.28. $ref
    5.29. $schema
6. Hyper Schema
    6.1. links
        6.1.1. Link Description Object
    6.2. fragmentResolution
        6.2.1. slash-delimited fragment resolution

6.2.2. dot-delimited fragment resolution
6.3. readonly
6.4. contentEncoding
6.5. pathStart
6.6. mediaType
7. Security Considerations
8. IANA Considerations
8.1. Registry of Link Relations
9. References
9.1. Normative References
9.2. Informative References
Appendix A. Change Log
Appendix B. Open Issues

---

1. Introduction

TOC

JSON (JavaScript Object Notation) Schema is a JSON media type for defining the structure of JSON data. JSON Schema provides a contract for what JSON data is required for a given application and how to interact with it. JSON Schema is intended to define validation, documentation, hyperlink navigation, and interaction control of JSON data.

---

2. Conventions

TOC

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 (Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," March 1997.) [RFC2119].

---

3. Overview

TOC

JSON Schema defines the media type "application/schema+json" for describing the structure of other JSON documents. JSON Schema is JSON-based and includes facilities for describing the structure of JSON documents in terms of allowable values, descriptions, and interpreting relations with other resources.
JSON Schema format is organized into several separate definitions. The first definition is the core schema specification. This definition is

primary concerned with describing a JSON structure and specifying valid elements in the structure. The second definition is the Hyper Schema specification which is intended define elements in a structure that can be interpreted as hyperlinks. Hyper Schema builds on JSON Schema to describe the hyperlink structure of other JSON documents and elements of interaction. This allows user agents to be able to successfully navigate JSON documents based on their schemas.
Cumulatively JSON Schema acts as a meta-document that can be used to define the required type and constraints on property values, as well as define the meaning of the property values for the purpose of describing a resource and determining hyperlinks within the representation.
An example JSON Schema that describes products might look like:

```
{
  "name":"Product",
  "properties":{
    "id":{
      "type":"number",
      "description":"Product identifier",
      "required":true
    },
    "name":{
      "description":"Name of the product",
      "type":"string",
      "required":true
    },
    "price":{
      "required":true,
      "type": "number",
      "minimum":0,
      "required":true
    },
    "tags":{
      "type":"array",
      "items":{
        "type":"string"
      }
    }
  },
  "links":[
    {
      "rel":"full",
      "href":"{id}"
    },
    {
      "rel":"comments",
      "href":"comments/?id={id}"
    }
  ]
}
```

This schema defines the properties of the instance JSON documents, the required properties (id, name, and price), as well as an optional property (tags). This also defines the link relations of the instance JSON documents.

TOC

3.1. Terminology

For this specification, **schema** will be used to denote a JSON Schema definition, and an **instance** refers to a JSON value that the schema will be describing and validating.

3.2. Design Considerations

TOC

The JSON Schema media type does not attempt to dictate the structure of JSON representations that contain data, but rather provides a separate format for flexibly communicating how a JSON representation should be interpreted and validated, such that user agents can properly understand acceptable structures and extrapolate hyperlink information with the JSON document. It is acknowledged that JSON documents come in a variety of structures, and JSON is unique in that the structure of stored data structures often prescribes a non-ambiguous definite JSON representation. Attempting to force a specific structure is generally not viable, and therefore JSON Schema allows for a great flexibility in the structure of the JSON data that it describes.
This specification is protocol agnostic. The underlying protocol (such as HTTP) should sufficiently define the semantics of the client-server interface, the retrieval of resource representations linked to by JSON representations, and modification of those resources. The goal of this format is to sufficiently describe JSON structures such that one can utilize existing information available in existing JSON representations from a large variety of services that leverage a representational state transfer architecture using existing protocols.

---

4. Schema/Instance Association

TOC

JSON Schema instances are correlated to their schema by the "describedby" relation, where the schema is defined to be the target of the relation. Instance representations may be of the "application/json" media type or any other subtype. Consequently, dictating how an instance representation should specify the relation to the schema is beyond the normative scope of this document (since this document specifically defines the JSON Schema media type, and no other), but it is recommended that instances specify their schema so that user agents can interpret the instance representation and messages may retain the self-descriptive characteristic, avoiding the need for out-of-band information about instance data. Two approaches are recommended for declaring the relation to the schema that describes the meaning of a JSON instance's (or collection of instances) structure. A MIME type

parameter named "profile" or a relation of "describedby" (which could be defined by a Link header) may be used:

```
Content-Type: application/my-media-type+json;
              profile=http://json.com/my-hyper-schema
```

or if the content is being transferred by a protocol (such as HTTP) that provides headers, a Link header can be used:

```
Link: <http://json.com/my-hyper-schema>; rel="describedby"
```

Instances MAY specify multiple schemas, to indicate all the schemas that are applicable to the data, and the data SHOULD be valid by all the schemas. The instance data MAY have multiple schemas that it is defined by (the instance data SHOULD be valid for those schemas). Or if the document is a collection of instances, the collection MAY contain instances from different schemas. When collections contain heterogeneous instances, the "pathStart" attribute MAY be specified in the schema to disambiguate which schema should be applied for each item in the collection. However, ultimately, the mechanism for referencing a schema is up to the media type of the instance documents (if they choose to specify that schemas can be referenced).

4.1. Self-Descriptive Schema

TOC

JSON Schemas can themselves be described using JSON Schemas. A self-describing JSON Schema for the core JSON Schema can be found at http://json-schema.org/schema for the latest version or http://json-schema.org/draft-03/schema for the draft-03 version. The hyper schema self-description can be found at http://json-schema.org/hyper-schema or http://json-schema.org/draft-03/hyper-schema. All schemas used within a protocol with media type definitions SHOULD include a MIME parameter that refers to the self-descriptive hyper schema or another schema that extends this hyper schema:

```
Content-Type: application/json;
              profile=http://json-schema.org/draft-03/hyper-schema
```

---

5. Core Schema Definition

TOC

A JSON Schema is a JSON Object that defines various attributes (including usage and valid values) of a JSON value. JSON Schema has

recursive capabilities; there are a number of elements in the structure that allow for nested JSON Schemas.
An example JSON Schema definition could look like:

```
{
  "description":"A person",
  "type":"object",

  "properties":{
    "name":{"type":"string"},
    "age" :{
        "type":"integer",
        "maximum":125
    }
  }
}
```

A JSON Schema object may have any of the following properties, called schema attributes (all attributes are optional):

5.1. type

TOC

This attribute defines what the primitive type or the schema of the instance MUST be in order to validate. This attribute can take one of two forms:

> **Simple Types** A string indicating a primitive or simple type. The following are acceptable string values:
>
> > **string** Value MUST be a string.
> >
> > **number** Value MUST be a number, floating point numbers are allowed.
> >
> > **integer** Value MUST be an integer, no floating point numbers are allowed. This is a subset of the number type.
> >
> > **boolean** Value MUST be a boolean.
> >
> > **object** Value MUST be an object.
> >
> > **array** Value MUST be an array.
> >
> > **null** Value MUST be null. Note this is mainly for purpose of being able use union types to define nullability. If this type is not included in a union, null values are not

allowed (the primitives listed above do not allow nulls on their own).

any Value MAY be of any type including null.

If the property is not defined or is not in this list, then any type of value is acceptable. Other type values MAY be used for custom purposes, but minimal validators of the specification implementation can allow any instance value on unknown type values.

Union Types An array of two or more simple type definitions. Each item in the array MUST be a simple type definition or a schema. The instance value is valid if it is of the same type as one of the simple type definitions, or valid by one of the schemas, in the array.

For example, a schema that defines if an instance can be a string or a number would be:

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

5.2. properties

TOC

This attribute is an object with property definitions that define the valid values of instance object property values. When the instance value is an object, the property values of the instance object MUST conform to the property definitions in this object. In this object, each property definition's value MUST be a schema, and the property's name MUST be the name of the instance property that it defines. The instance property value MUST be valid according to the schema from the property definition. Properties are considered unordered, the order of the instance properties MAY be in any order.

5.3. patternProperties

TOC

This attribute is an object that defines the schema for a set of property names of an object instance. The name of each property of this attribute's object is a regular expression pattern in the ECMA 262/Perl 5 format, while the value is a schema. If the pattern matches the name of a property on the instance object, the value of the instance's property MUST be valid against the pattern name's schema value.

5.4. additionalProperties

TOC

This attribute defines a schema for all properties that are not explicitly defined in an object type definition. If specified, the value MUST be a schema or a boolean. If false is provided, no additional properties are allowed beyond the properties defined in the schema. The default value is an empty schema which allows any value for additional properties.

5.5. items

TOC

This attribute defines the allowed items in an instance array, and MUST be a schema or an array of schemas. The default value is an empty schema which allows any value for items in the instance array.
When this attribute value is a schema and the instance value is an array, then all the items in the array MUST be valid according to the schema.
When this attribute value is an array of schemas and the instance value is an array, each position in the instance array MUST conform to the schema in the corresponding position for this array. This called tuple typing. When tuple typing is used, additional items are allowed, disallowed, or constrained by the "additionalItems" (additionalItems) attribute using the same rules as "additionalProperties" (additionalProperties) for objects.

5.6. additionalItems

TOC

This provides a definition for additional items in an array instance when tuple definitions of the items is provided. This can be false to indicate additional items in the array are not allowed, or it can be a schema that defines the schema of the additional items.

5.7. required

TOC

This attribute indicates if the instance must have a value, and not be undefined. This is false by default, making the instance optional.

TOC

5.8. dependencies

This attribute is an object that defines the requirements of a property on an instance object. If an object instance has a property with the same name as a property in this attribute's object, then the instance must be valid against the attribute's property value (hereafter referred to as the "dependency value").
The dependency value can take one of two forms:

> **Simple Dependency** If the dependency value is a string, then the instance object MUST have a property with the same name as the dependency value. If the dependency value is an array of strings, then the instance object MUST have a property with the same name as each string in the dependency value's array.
>
> **Schema Dependency** If the dependency value is a schema, then the instance object MUST be valid against the schema.

5.9. minimum

TOC

This attribute defines the minimum value of the instance property when the type of the instance value is a number.

5.10. maximum

TOC

This attribute defines the maximum value of the instance property when the type of the instance value is a number.

5.11. exclusiveMinimum

TOC

This attribute indicates if the value of the instance (if the instance is a number) can not equal the number defined by the "minimum" attribute. This is false by default, meaning the instance value can be greater then or equal to the minimum value.

TOC

5.12. exclusiveMaximum

This attribute indicates if the value of the instance (if the instance is a number) can not equal the number defined by the "maximum" attribute. This is false by default, meaning the instance value can be less then or equal to the maximum value.

5.13. minItems

TOC

This attribute defines the minimum number of values in an array when the array is the instance value.

5.14. maxItems

TOC

This attribute defines the maximum number of values in an array when the array is the instance value.

5.15. uniqueItems

TOC

This attribute indicates that all items in an array instance MUST be unique (contains no two identical values).
Two instance are consider equal if they are both of the same type and:

> are null; or
>
> are booleans/numbers/strings and have the same value; or
>
> are arrays, contains the same number of items, and each item in the array is equal to the corresponding item in the other array; or
>
> are objects, contains the same property names, and each property in the object is equal to the corresponding property in the other object.

5.16. pattern

TOC

When the instance value is a string, this provides a regular expression that a string instance MUST match in order to be valid. Regular

expressions SHOULD follow the regular expression specification from ECMA 262/Perl 5

5.17. minLength

TOC

When the instance value is a string, this defines the minimum length of the string.

5.18. maxLength

TOC

When the instance value is a string, this defines the maximum length of the string.

5.19. enum

TOC

This provides an enumeration of all possible values that are valid for the instance property. This MUST be an array, and each item in the array represents a possible value for the instance value. If this attribute is defined, the instance value MUST be one of the values in the array in order for the schema to be valid. Comparison of enum values uses the same algorithm as defined in "uniqueItems" (uniqueItems).

5.20. default

TOC

This attribute defines the default value of the instance when the instance is undefined.

5.21. title

TOC

This attribute is a string that provides a short description of the instance property.

TOC

5.22. description

This attribute is a string that provides a full description of the of purpose the instance property.

5.23. format

TOC

This property defines the type of data, content type, or microformat to be expected in the instance property values. A format attribute MAY be one of the values listed below, and if so, SHOULD adhere to the semantics describing for the format. A format SHOULD only be used to give meaning to primitive types (string, integer, number, or boolean). Validators MAY (but are not required to) validate that the instance values conform to a format. The following formats are predefined:

date-time This SHOULD be a date in ISO 8601 format of YYYY-MM-DDThh:mm:ssZ in UTC time. This is the recommended form of date/timestamp.

date This SHOULD be a date in the format of YYYY-MM-DD. It is recommended that you use the "date-time" format instead of "date" unless you need to transfer only the date part.

time This SHOULD be a time in the format of hh:mm:ss. It is recommended that you use the "date-time" format instead of "time" unless you need to transfer only the time part.

utc-millisec This SHOULD be the difference, measured in milliseconds, between the specified time and midnight, 00:00 of January 1, 1970 UTC. The value SHOULD be a number (integer or float).

regex A regular expression, following the regular expression specification from ECMA 262/Perl 5.

color This is a CSS color (like "#FF0000" or "red"), based on CSS 2.1 (Hickson, I., Lie, H., Çelik, T., and B. Bos, “Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification,” July 2007.) [W3C.CR-CSS21-20070719].

style This is a CSS style definition (like "color: red; background-color:#FFF"), based on CSS 2.1 (Hickson, I., Lie, H., Çelik, T., and B. Bos, “Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification,” July 2007.) [W3C.CR-CSS21-20070719].

phone This SHOULD be a phone number (format MAY follow E.123).

uri
: This value SHOULD be a URI..

email This SHOULD be an email address.

ip-address This SHOULD be an ip version 4 address.

ipv6 This SHOULD be an ip version 6 address.

host-name This SHOULD be a host-name.

Additional custom formats MAY be created. These custom formats MAY be expressed as an URI, and this URI MAY reference a schema of that format.

5.24. divisibleBy

TOC

This attribute defines what value the number instance must be divisible by with no remainder (the result of the division must be an integer.) The value of this attribute SHOULD NOT be 0.

5.25. disallow

TOC

This attribute takes the same values as the "type" attribute, however if the instance matches the type or if this value is an array and the instance matches any type or schema in the array, then this instance is not valid.

5.26. extends

TOC

The value of this property MUST be another schema which will provide a base schema which the current schema will inherit from. The inheritance rules are such that any instance that is valid according to the current schema MUST be valid according to the referenced schema. This MAY also be an array, in which case, the instance MUST be valid for all the schemas in the array. A schema that extends another schema MAY define additional attributes, constrain existing attributes, or add other constraints.
Conceptually, the behavior of extends can be seen as validating an instance against all constraints in the extending schema as well as the extended schema(s). More optimized implementations that merge schemas are possible, but are not required. An example of using "extends":

```
{
  "description":"An adult",
  "properties":{"age":{"minimum": 21}},
  "extends":"person"
}
```

```
{
  "description":"Extended schema",
  "properties":{"deprecated":{"type": "boolean"}},
  "extends":"http://json-schema.org/draft-03/schema"
}
```

5.27. id

TOC

This attribute defines the current URI of this schema (this attribute is effectively a "self" link). This URI MAY be relative or absolute. If the URI is relative it is resolved against the current URI of the parent schema it is contained in. If this schema is not contained in any parent schema, the current URI of the parent schema is held to be the URI under which this schema was addressed. If id is missing, the current URI of a schema is defined to be that of the parent schema. The current URI of the schema is also used to construct relative references such as for $ref.

5.28. $ref

TOC

This attribute defines a URI of a schema that contains the full representation of this schema. When a validator encounters this attribute, it SHOULD replace the current schema with the schema referenced by the value's URI (if known and available) and re-validate the instance. This URI MAY be relative or absolute, and relative URIs SHOULD be resolved against the URI of the current schema.

5.29. $schema

TOC

This attribute defines a URI of a JSON Schema that is the schema of the current schema. When this attribute is defined, a validator SHOULD use the schema referenced by the value's URI (if known and available) when resolving Hyper Schema (Hyper Schema) links (links).

A validator MAY use this attribute's value to determine which version of JSON Schema the current schema is written in, and provide the appropriate validation features and behavior. Therefore, it is RECOMMENDED that all schema authors include this attribute in their schemas to prevent conflicts with future JSON Schema specification changes.

---

6. Hyper Schema

TOC

The following attributes are specified in addition to those attributes that already provided by the core schema with the specific purpose of informing user agents of relations between resources based on JSON data. Just as with JSON schema attributes, all the attributes in hyper schemas are optional. Therefore, an empty object is a valid (non-informative) schema, and essentially describes plain JSON (no constraints on the structures). Addition of attributes provides additive information for user agents.

6.1. links

TOC

The value of the links property MUST be an array, where each item in the array is a link description object which describes the link relations of the instances.

6.1.1. Link Description Object

TOC

A link description object is used to describe link relations. In the context of a schema, it defines the link relations of the instances of the schema, and can be parameterized by the instance values. The link description format can be used on its own in regular (non-schema documents), and use of this format can be declared by referencing the normative link description schema as the the schema for the data structure that uses the links. The URI of the normative link description schema is: http://json-schema.org/links (latest version) or http://json-schema.org/draft-03/links (draft-03 version).

TOC

6.1.1.1. href

The value of the "href" link description property indicates the target URI of the related resource. The value of the instance property SHOULD be resolved as a URI-Reference per RFC 3986 (Berners-Lee, T., Fielding, R., and L. Masinter, “Uniform Resource Identifier (URI): Generic Syntax,” January 2005.) [RFC3986] and MAY be a relative URI. The base URI to be used for relative resolution SHOULD be the URI used to retrieve the instance object (not the schema) when used within a schema. Also, when links are used within a schema, the URI SHOULD be parametrized by the property values of the instance object, if property values exist for the corresponding variables in the template (otherwise they MAY be provided from alternate sources, like user input).
Instance property values SHOULD be substituted into the URIs where matching braces ('{', '}') are found surrounding zero or more characters, creating an expanded URI. Instance property value substitutions are resolved by using the text between the braces to denote the property name from the instance to get the value to substitute. For example, if an href value is defined:

```
http://somesite.com/{id}
```

Then it would be resolved by replace the value of the "id" property value from the instance object. If the value of the "id" property was "45", the expanded URI would be:

```
http://somesite.com/45
```

If matching braces are found with the string "@" (no quotes) between the braces, then the actual instance value SHOULD be used to replace the braces, rather than a property value. This should only be used in situations where the instance is a scalar (string, boolean, or number), and not for objects or arrays.

6.1.1.2. rel

TOC

The value of the "rel" property indicates the name of the relation to the target resource. The relation to the target SHOULD be interpreted as specifically from the instance object that the schema (or sub-schema) applies to, not just the top level resource that contains the object within its hierarchy. If a resource JSON representation contains a sub object with a property interpreted as a link, that sub-object holds the relation with the target. A relation to target from the top level resource MUST be indicated with the schema describing the top level JSON representation.
Relationship definitions SHOULD NOT be media type dependent, and users are encouraged to utilize existing accepted relation definitions,

including those in existing relation registries (see RFC 4287 (Nottingham, M., Ed. and R. Sayre, Ed., “The Atom Syndication Format,” December 2005.) [RFC4287]). However, we define these relations here for clarity of normative interpretation within the context of JSON hyper schema defined relations:

> **self** If the relation value is "self", when this property is encountered in the instance object, the object represents a resource and the instance object is treated as a full representation of the target resource identified by the specified URI.
>
> **full** This indicates that the target of the link is the full representation for the instance object. The object that contains this link possibly may not be the full representation.
>
> **describedby** This indicates the target of the link is the schema for the instance object. This MAY be used to specifically denote the schemas of objects within a JSON object hierarchy, facilitating polymorphic type data structures.
>
> **root** This relation indicates that the target of the link SHOULD be treated as the root or the body of the representation for the purposes of user agent interaction or fragment resolution. All other properties of the instance objects can be regarded as meta-data descriptions for the data.

The following relations are applicable for schemas (the schema as the "from" resource in the relation):

> **instances** This indicates the target resource that represents collection of instances of a schema.
>
> **create** This indicates a target to use for creating new instances of a schema. This link definition SHOULD be a submission link with a non-safe method (like POST).

For example, if a schema is defined:

```
{
  "links": [
    {
      "rel": "self"
      "href": "{id}"
    },
    {
      "rel": "up"
      "href": "{upId}"
    },
    {
      "rel": "children"
      "href": "?upId={id}"
    }
  ]
}
```

And if a collection of instance resource's JSON representation was retrieved:

```
GET /Resource/

[
  {
    "id": "thing",
    "upId": "parent"
  },
  {
    "id": "thing2",
    "upId": "parent"
  }
]
```

This would indicate that for the first item in the collection, its own (self) URI would resolve to "/Resource/thing" and the first item's "up" relation SHOULD be resolved to the resource at "/Resource/parent". The "children" collection would be located at "/Resource/?upId=thing".

6.1.1.3. targetSchema

TOC

This property value is a schema that defines the expected structure of the JSON representation of the target of the link.

TOC

6.1.1.4. Submission Link Properties

The following properties also apply to link definition objects, and provide functionality analogous to HTML forms, in providing a means for submitting extra (often user supplied) information to send to a server.

6.1.1.4.1. method

TOC

This attribute defines which method can be used to access the target resource. In an HTTP environment, this would be "GET" or "POST" (other HTTP methods such as "PUT" and "DELETE" have semantics that are clearly implied by accessed resources, and do not need to be defined here). This defaults to "GET".

6.1.1.4.2. enctype

TOC

If present, this property indicates a query media type format that the server supports for querying or posting to the collection of instances at the target resource. The query can be suffixed to the target URI to query the collection with property-based constraints on the resources that SHOULD be returned from the server or used to post data to the resource (depending on the method). For example, with the following schema:

```
{
 "links":[
   {
     "enctype":"application/x-www-form-urlencoded",
     "method":"GET",
     "href":"/Product/",
     "properties":{
        "name":{"description":"name of the product"}
     }
   }
 ]
}
```

This indicates that the client can query the server for instances that have a specific name:

```
/Product/?name=Slinky
```

If no enctype or method is specified, only the single URI specified by the href property is defined. If the method is POST, "application/json" is the default media type.

6.1.1.4.3. schema

TOC

This attribute contains a schema which defines the acceptable structure of the submitted request (for a GET request, this schema would define the properties for the query string and for a POST request, this would define the body).

6.2. fragmentResolution

TOC

This property indicates the fragment resolution protocol to use for resolving fragment identifiers in URIs within the instance representations. This applies to the instance object URIs and all children of the instance object's URIs. The default fragment resolution protocol is "slash-delimited", which is defined below. Other fragment resolution protocols MAY be used, but are not defined in this document. The fragment identifier is based on RFC 2396, Sec 5 (Berners-Lee, T., Fielding, R., and L. Masinter, “Uniform Resource Identifiers (URI): Generic Syntax,” August 1998.) [RFC2396], and defines the mechanism for resolving references to entities within a document.

6.2.1. slash-delimited fragment resolution

TOC

With the slash-delimited fragment resolution protocol, the fragment identifier is interpreted as a series of property reference tokens that start with and are delimited by the "/" character (\x2F). Each property reference token is a series of unreserved or escaped URI characters. Each property reference token SHOULD be interpreted, starting from the beginning of the fragment identifier, as a path reference in the target JSON structure. The final target value of the fragment can be determined by starting with the root of the JSON structure from the representation of the resource identified by the pre-fragment URI. If the target is a JSON object, then the new target is the value of the property with the name identified by the next property reference token in the fragment. If the target is a JSON array, then the target is determined by finding the item in array the array with the index defined by the next property reference token (which MUST be a number).

The target is successively updated for each property reference token, until the entire fragment has been traversed.
Property names SHOULD be URI-encoded. In particular, any "/" in a property name MUST be encoded to avoid being interpreted as a property delimiter.
For example, for the following JSON representation:

```
{
  "foo":{
    "anArray":[
      {"prop":44}
    ],
    "another prop":{
      "baz":"A string"
    }
  }
}
```

The following fragment identifiers would be resolved:

fragment identifier	resolution
#	self, the root of the resource itself
#/foo	the object referred to by the foo property
#/foo/another%20prop	the object referred to by the "another prop" property of the object referred to by the "foo" property
#/foo/another%20prop/baz	the string referred to by the value of "baz" property of the "another prop" property of the object referred to by the "foo" property
#/foo/anArray/0	the first object in the "anArray" array

6.2.2. dot-delimited fragment resolution

TOC

The dot-delimited fragment resolution protocol is the same as slash-delimited fragment resolution protocol except that the "." character (\x2E) is used as the delimiter between property names (instead of "/") and the path does not need to start with a ".". For example, #.foo and #foo are a valid fragment identifiers for referencing the value of the foo propery.

TOC

6.3. readonly

This attribute indicates that the instance property SHOULD NOT be changed. Attempts by a user agent to modify the value of this property are expected to be rejected by a server.

6.4. contentEncoding

TOC

If the instance property value is a string, this attribute defines that the string SHOULD be interpreted as binary data and decoded using the encoding named by this schema property. RFC 2045, Sec 6.1 (Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies," November 1996.) [RFC2045] lists the possible values for this property.

6.5. pathStart

TOC

This attribute is a URI that defines what the instance's URI MUST start with in order to validate. The value of the "pathStart" attribute MUST be resolved as per RFC 3986, Sec 5 (Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax," January 2005.) [RFC3986], and is relative to the instance's URI. When multiple schemas have been referenced for an instance, the user agent can determine if this schema is applicable for a particular instance by determining if the URI of the instance begins with the the value of the "pathStart" attribute. If the URI of the instance does not start with this URI, or if another schema specifies a starting URI that is longer and also matches the instance, this schema SHOULD NOT be applied to the instance. Any schema that does not have a pathStart attribute SHOULD be considered applicable to all the instances for which it is referenced.

6.6. mediaType

TOC

This attribute defines the media type of the instance representations that this schema is defining.

TOC

7. Security Considerations

This specification is a sub-type of the JSON format, and consequently the security considerations are generally the same as RFC 4627 (Crockford, D., “The application/json Media Type for JavaScript Object Notation (JSON),” July 2006.) [RFC4627]. However, an additional issue is that when link relation of "self" is used to denote a full representation of an object, the user agent SHOULD NOT consider the representation to be the authoritative representation of the resource denoted by the target URI if the target URI is not equivalent to or a sub-path of the the URI used to request the resource representation which contains the target URI with the "self" link. For example, if a hyper schema was defined:

```
{
  "links":[
        {
          "rel":"self",
          "href":"{id}"
        }
  ]
}
```

And a resource was requested from somesite.com:

```
GET /foo/
```

With a response of:

```
Content-Type: application/json; profile=/schema-for-this-data
[
  {"id":"bar", "name":"This representation can be safely treated \
        as authoritative "},
  {"id":"/baz", "name":"This representation should not be treated as \
        authoritative the user agent should make request the resource\
        from "/baz" to ensure it has the authoritative representation"},
  {"id":"http://othersite.com/something", "name":"This representation\
        should also not be treated as authoritative and the target\
        resource representation should be retrieved for the\
        authoritative representation"}
]
```

---

8. IANA Considerations

TOC

The proposed MIME media type for JSON Schema is "application/schema+json".

Type name: application
Subtype name: schema+json
Required parameters: profile
The value of the profile parameter SHOULD be a URI (relative or absolute) that refers to the schema used to define the structure of this structure (the meta-schema). Normally the value would be http://json-schema.org/draft-03/hyper-schema, but it is allowable to use other schemas that extend the hyper schema's meta- schema.
Optional parameters: pretty
The value of the pretty parameter MAY be true or false to indicate if additional whitespace has been included to make the JSON representation easier to read.

8.1. Registry of Link Relations

TOC

This registry is maintained by IANA per RFC 4287 (Nottingham, M., Ed. and R. Sayre, Ed., "The Atom Syndication Format," December 2005.) [RFC4287] and this specification adds four values: "full", "create", "instances", "root". New assignments are subject to IESG Approval, as outlined in RFC 5226 (Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs," May 2008.) [RFC5226]. Requests should be made by email to IANA, which will then forward the request to the IESG, requesting approval.

---

9. References

TOC

9.1. Normative References

TOC

[RFC2045]	Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies," RFC 2045, November 1996 (TXT).
[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).
[RFC2396]	Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," RFC 2396, August 1998 (TXT, HTML, XML).
[RFC3339]	Klyne, G., Ed. and C. Newman, "Date and Time on the Internet: Timestamps," RFC 3339, July 2002 (TXT, HTML, XML).
[RFC3986]	

	Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax," STD 66, RFC 3986, January 2005 (TXT, HTML, XML).
[RFC4287]	Nottingham, M., Ed. and R. Sayre, Ed., "The Atom Syndication Format," RFC 4287, December 2005 (TXT, HTML, XML).

9.2. Informative References

TOC

[RFC2616]	Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1," RFC 2616, June 1999 (TXT, PS, PDF, HTML, XML).
[RFC4627]	Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)," RFC 4627, July 2006 (TXT).
[RFC5226]	Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs," BCP 26, RFC 5226, May 2008 (TXT).
[I-D.hammer-discovery]	Hammer-Lahav, E., "LRDD: Link-based Resource Descriptor Discovery," draft-hammer-discovery-06 (work in progress), May 2010 (TXT).
[I-D.gregorio-uritemplate]	Gregorio, J., Fielding, R., Hadley, M., and M. Nottingham, "URI Template," draft-gregorio-uritemplate-04 (work in progress), March 2010 (TXT).
[I-D.nottingham-http-link-header]	Nottingham, M., "Web Linking," draft-nottingham-http-link-header-10 (work in progress), May 2010 (TXT).
[W3C.REC-html401-19991224]	Raggett, D., Hors, A., and I. Jacobs, "HTML 4.01 Specification," World Wide Web Consortium Recommendation REC-html401-19991224, December 1999 (HTML).
[W3C.CR-CSS21-20070719]	Hickson, I., Lie, H., Çelik, T., and B. Bos, "Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification," World Wide Web Consortium CR CR-CSS21-20070719, July 2007 (HTML).

Appendix A. Change Log

TOC

draft-03* Added example and verbiage to "extends" attribute.

*Defined slash-delimited to use a leading slash.

*Made "root" a relation instead of an attribute.

*Removed address values, and MIME media type from format to reduce confusion (mediaType already exists, so it can be used for MIME types).

*Added more explanation of nullability.

*Removed "alternate" attribute.

*Upper cased many normative usages of must, may, and should.

*Replaced the link submission "properties" attribute to "schema" attribute.

*Replaced "optional" attribute with "required" attribute.

*Replaced "maximumCanEqual" attribute with "exclusiveMaximum" attribute.

*Replaced "minimumCanEqual" attribute with "exclusiveMinimum" attribute.

*Replaced "requires" attribute with "dependencies" attribute.

*Moved "contentEncoding" attribute to hyper schema.

*Added "additionalItems" attribute.

*Added "id" attribute.

*Switched self-referencing variable substitution from "-this" to "@" to align with reserved characters in URI template.

*Added "patternProperties" attribute.

*Schema URIs are now namespace versioned.

*Added "$ref" and "$schema" attributes.

draft-02 * Replaced "maxDecimal" attribute with "divisibleBy" attribute.

*Added slash-delimited fragment resolution protocol and made it the default.

*Added language about using links outside of schemas by referencing its normative URI.

*Added "uniqueItems" attribute.

*Added "targetSchema" attribute to link description object.

draft-01* Fixed category and updates from template.

draft-00* Initial draft.

Appendix B. Open Issues

TOC

Should we give a preference to MIME headers over Link headers (or only use one)?

Should "root" be a MIME parameter?

Should "format" be renamed to "mediaType" or "contentType" to reflect the usage MIME media types that are allowed?

How should dates be handled?

Authors' Addresses

TOC

	Kris Zyp (editor)
	SitePen (USA)
	530 Lytton Avenue
	Palo Alto, CA 94301
	USA
Phone:	+1 650 968 8787
EMail:	kris@sitepen.com
	Gary Court
	Calgary, AB
	Canada
EMail:	gary.court@gmail.com