**Document Structure**

By:

Anu Shivhare

David Welles

Franklyn Reyes

Ibtxhis Khang

Nicholas Harrison

Useful Links

<https://www.youtube.com/watch?v=cuEtnrL9-H0>

<https://www.youtube.com/watch?v=iiADhChRriM>

<https://jsonapi.org/recommendations/>

**Meta information**:

· A meta member can be used to include non-standard meta information.

· The value of each “meta” member must be an object.

· Any members may be specified with “meta” object



**Links**:

* Link members can be used to represent links.
* Value of each “links” member must be an object.
* Each member of the links object must be a link which should either be
  + String containing the link’s URL
  + An object containing
    - **href:** string containing link’s URL
    - **meta:** meta object containing non-standard meta information about the link

**Example:**

* The following “self” link is a simple URL



* The below link include URL as well as meta information:



**DOCUMENT STRUCTURE**

**Top Level:**

A document **MUST** contain at least one of the following top-level members:

1. Data: the document’s “primary data”

- Valid data types include strings, numbers, objects, arrays, booleans, and null.

Examples:

Number - { "age":30 }

Object - {"employee":{ "name":"John", "age":30, "city":"New York" } }

Array - {"employees":[ "John", "Anna", "Peter" ] }

Boolean - { "sale":true }

Null - { "middlename":null }

- Invalid data types include functions, dates, and undefined.

1. Errors: an array of error objects
2. Meta: a meta object that contains non-standard meta-information.

(The members data and errors MUST NOT coexist in the same document.)

**Resource objects:**

The main purpose of resource objects is to represent resources inside a JSON document and are surrounded by curly braces and hold key value pairs. Example:

{

"key": "value", (All key value pairs require seperation

"key": "value" by a comma except the final pair.)

}

A resource object has specific members it must contain and members that it may contain:

1. **Must** contain - id & type

2. May contain

* Attributes - represents resources data
* Relationships - object describing relationship between the resource and other API resources
* links - any links related to the resource
* meta - information about a resource that can not be represented as an attribute or relationship

Example:

{

"type": "articles",

"id": "1",

"attributes": {

"title": "Rails is Omakase"

},

"relationships": {

"author": {

"links": {

"self": "/articles/1/relationships/author",

"related": "/articles/1/author"

},

"data": { "type": "people", "id": "9" }

}

}

}

**Identification:**

Every resource object **MUST** contain an id member and a type member. The values of the id and type members **MUST** be strings and identify a single, unique resource.

Type Members:

1. Used to describe resource objects that share common attributes and relationships.
2. The values **Must** adhere to the same constraints as member names.

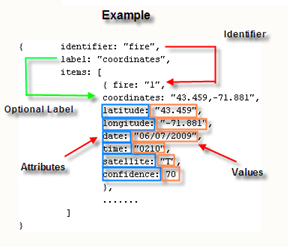
**Fields:**

A resource object’s attributes and its relationships are collectively called its “fields”. Fields **Must** share a common namespace with each other and with type and id. In other words, a resource can not have an attribute and relationship with the same name, nor can it have an attribute or relationship named type or id.

**Attributes**

Attributes must have a value of an object. Members of the attribute object represent information about the resource object in which it’s defined.

* Attributes may contain any valid JSON value



Complex data structures involving JSON objects and arrays are allowed as attribute values. But any object that is a part of or is contained in an attribute **MUST NOT** contain a relationships or links member, those are reserved for future use only.

Although has-one foreign keys (e.g. author\_id) are often stored internally alongside other information to be represented in a *resource object 1*, these keys **SHOULD NOT** appear as attributes.

1. ***Remember a resource object is written in key:value pairings!***

* *Keys must be strings*
* *Values can be strings, numbers, objects, arrays, booleans, or null*

**Relationships:**

must contain at least one of the following:

self: a link for the relationship itself ('Relationship Link'). Allows the client to manipulate the relationship. EX: removing an author through an article's relationship URL would disconnect the person from the article without deleting the people resource itself. In other words it only removes the related id or content you are trying to manipulate.

{

"relationships": {

"author": {

"links": {

"self": "http://example.com/articles/1/relationships/author",

"related": "http://example.com/articles/1/author"

},

"data": { "type": "people", "id": "9" }

}

},

"links": {

"self": "http://example.com/articles/1"

}

}

**Related Resource Links:** Example:

The main objective is to provide access to resource objects linked in a relationship. When fetched, the related resource objects are returned as the response's primary data. For example when you comment an article it may return another related collection of comments through GET requests. It is similar to an HTML hyperlink text where you click and it redirects you to another source or webpage. RRL must contain a valid URL for it to work properly. Also a related resource link MUST NOT change if its relationship content changes.

**RESOURCE LINKAGE:**

Resource linkage in a compound document allows a client to link together all of the included resource objects without having to GET any URLs via links.

Resource linkage MUST be represented as one of the following:

null for empty to-one relationships.

an empty array ([]) for empty to-many relationships.

a single resource identifier object for non-empty to-one relationships.

an array of resource identifier objects for non-empty to-many relationships.

Examples:

{

"type": "articles",

"id": "1",

"attributes": {

"title": "Rails is Omakase"

},

"relationships": {

"author": {

"links": {

"self": "http://example.com/articles/1/relationships/author",

"related": "http://example.com/articles/1/author"

},

"data": { "type": "people", "id": "9" }

}

},

"links": {

"self": "http://example.com/articles/1"

}

}

Example 2:

ID member and type member

For this schema, each object's relationships are defined as an object under

a "links" property.

Each property of that object is the name of the relationship, and the value

is a single object with the "type" and "id" of the related object.

For "to-many" relationships, the value can be an array of the same objects.

\*/

var data = {

"articles": [{

"id": "article1",

"links": {

"author": {"id": "person1", "type": "people"}

}

The author relationship includes a link for the relationship itself (which allows the client to change the related author directly), a related resource link to fetch the resource objects, and linkage information.

**RESOURCE LINKS:**

The author relationship includes a link for the relationship itself (which allows the client to change the related author directly), a related resource link to fetch the resource objects, and linkage information.

**Resource Links**

The optional links member within each resource object contains links related to the resource.

If present, this links object MAY contain a self link that identifies the resource represented by the resource object.

{

"type": "articles",

"id": "1",

"attributes": {

"title": "Rails is Omakase"

},

"links": {

"self": "http://example.com/articles/1"

}

}

A server MUST respond to a GET request to the specified URL with a response that includes the resource as the primary data.

**JSON: API OBJECT**

A JSON Object occurs when a JSON:API document includes information under a top-level JSONAPI member such as: data, errors, meta, jsonapi, links, and included.

A JSONAPI may contain a string indicating the highest JSON API version which could contain a meta member.

Meta member has a value of a meta object that contains non-standard meta-information.

Example:

|  |
| --- |
| {  "jsonapi": {  "version": "1.0"  }  } |

→ If the version member is not present, clients should assume the server is at least the 1.0 version specification.

→ JSON: API is committed to making changes, the version member indicates which new feature the server may support.

**MEMBER NAMES**

- All member names must be treated as case-sensitive by clients and servers and must follow the following conditions:  
 1. Member names must contain at least 1 character.

2. Member names must contain only the allowed characters listed below.

3. Member names must start and end with a “globally allowed character”

*Recommended*: Member names use only non-reserved (-, \_), URL safe characters and not use characters that are reserved such as ? etc.

\*\* Naming is important when mixing profiles authored by different parties. \*\*

\*\* NOTE: U+0061 → is the unicode java source code for a.

Allowed Characters:

*Globally Allowed Characters* (may be used anywhere in a member name)

- U+0061 to U+007A **“end in any letters a-z”**

- U+0041 to U+005A, **“contain any letters A-Z”**

- U+0030 to U+0039, **“contain any numbers 0-9”**

- U+0080 and above (non-ASCII Unicode Characters → not recommended nor URL safe)

Also, the following characters can be used only as the first or last characters in a member name.

U+002D **HYPHEN-MINUS, “-“**

U+005F **LOW LINE, “\_”**

U+0020 **SPACE, “ “ (not recommended, not URL safe)**

Reserved Characters:

The following are *not to* be used in member names:

|  |  |
| --- | --- |
| U+002B **PLUS SIGN, “+”** (used for ordering)  U+002C **COMMA, “,”** (used as a separator between relationship paths)  U+002E **PERIOD, “.”** (used as a separator within relationship paths)  U+005B **LEFT SQUARE BRACKET,** “[” (used in sparse fieldsets)  U+005D **RIGHT SQUARE BRACKET,** “]” (used in sparse fieldsets)  U+0021 **EXCLAMATION MARK, “!”**  U+0022 **QUOTATION MARK, ‘”’**  U+0023 **NUMBER SIGN, “#”**  U+0024 **DOLLAR SIGN, “$”**  U+0025 **PERCENT SIGN, “%”**  U+0026 **AMPERSAND, “&”**  U+0027 **APOSTROPHE, “’”**  U+0028 **LEFT PARENTHESIS, “(“**  U+0029 **RIGHT PARENTHESIS, “)”** | U+002A **ASTERISK, “\*”**  U+002F **SOLIDUS, “/**”  U+003A **COLON, “:”**  U+003B **SEMICOLON, “;”**  U+003C **LESS-THAN SIGN, “<”**  U+003D **EQUALS SIGN, “=”**  U+003E **GREATER-THAN SIGN, “>”**  U+003F **QUESTION MARK, “?”**  U+0040 **COMMERCIAL AT, “@”**  U+005C **REVERSE SOLIDUS, “\”**  U+005E **CIRCUMFLEX ACCENT, “^”**  U+0060 **GRAVE ACCENT, “`”**  U+007B **LEFT CURLY BRACKET, “{“**  U+007C **VERTICAL LINE, “|”**  U+007D **RIGHT CURLY BRACKET, “}”**  U+007E **TILDE, “~”**  U+007F **DELETE**  U+0000 to U+001F **(C0 Controls)** |

**Resource Identifier Objects** are objects that identify an individual resource.

* Must contain a TYPE and ID member
* Could also include a META member, which means that that meta member has a object as a value and its non-standard meta information

## Compound Documents:

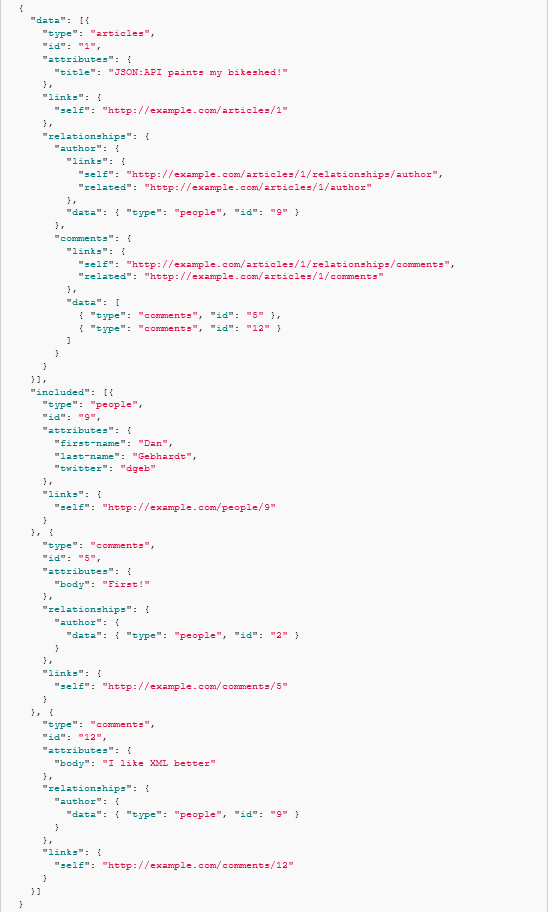
**Compound Documents** happen when a server allows responses that include related resources along with the requested primary resources, this happens to reduce the number of HTTP requests.

* ALL included resources MUST be represented as an array of resource objects in a top-level included member
* A compound document MUST NOT have more than one resource object for each type and ID pair!

Compound documents require *full linkage*:

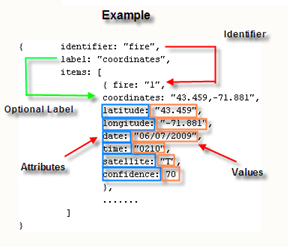
Full linkage just means that every included resource has at least one resource identifier object to identify it in the same document. The RIO could be either primary data or just representing resource linkage



****

**Attributes** must have a value of an object. Members of the attribute object represent information about the resource object in which it’s defined.

* Attributes may contain any valid JSON value



Complex data structures involving JSON objects and arrays are allowed as attribute values. But any object that is a part of or is contained in an attribute **MUST NOT** contain a relationships or links member, those are reserved for future use only.

Although has-one foreign keys (e.g. author\_id) are often stored internally alongside other information to be represented in a *resource object 1*, these keys **SHOULD NOT** appear as attributes.

1. ***Remember a resource object is written in key:value pairings!***

* *Keys must be strings*
* *Values can be strings, numbers, objects, arrays, booleans, or null*