Internet Engineering Task Force   P. Bryan, Ed.
Internet-Draft   Salesforce.com
Intended status: Informational   August 6, 2012
Expires: February 7, 2013

A Convention for HTTP Access to JSON Resources
draft-pbryan-http-json-resource-03

Abstract

A convention for accessing JSON representations of resources via HTTP, promoting a uniform interface across multiple resources and reuse of software components.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

JavaScript Object Notation (JSON) [RFC4627] is a common format for the representation of structured data. Hypertext Transfer Protocol (HTTP) [RFC2616] is the standard protocol for providing access to resources.

This document codifies a convention for accessing JSON representations of resources via HTTP. This promotes a uniform interface across multiple resources and reuse of conforming server and client software components.

2. Conventions

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 [RFC2119].

This document expresses the structure of Request-URIs in examples using URI Template [RFC6570] syntax.

3. Resources

A resource accessed through this convention is represented as a JSON value with an "application/json" (or derivative) Internet media type.

An accessible resource is a member of a collection of resources. A collection of resources MUST have a unique location, which is expressed as a part of the Request-URI of HTTP requests. The server implementation determines the location of a resource collection.

Each resource MUST have a unique identifier within a collection. The server implementation MAY establish restrictions on what identifiers can be used, and reject requests that have identifiers that do not conform with a 403 Forbidden status code.

4. Version Control

A server MAY implement version control for resources, and use it as the basis of an optimistic concurrency control mechanism. If version control is implemented for a given resource, the server MUST expose the resource version in responses, and clients SHOULD use preconditions when performing operations that modify such resources.

The server expresses the resource version in an "ETag" response

header, and the "_rev" metadata member in JSON object entity responses. The client expresses resource version in the "If-Match" and "If-None-Match" precondition request headers.

If a server implements version control for a resource, and the version specified by a client in a request does not match the current version of the resource, then the server SHOULD respond with a 412 Precondition Failed error status code.

If a server implements version control for a resource and the resource version is not specified by a client for an operation that modifies the resource, the server MAY allow the operation, or reject it with a 428 Precondition Required status code, per [RFC6585].

The server implementation determines how a resource version is computed; it MUST ensure that different values of a given resource results in different versions. Clients SHOULD treat a resource version provided by a server as opaque.

5. Operations

This convention provides a uniform set of operations, all of which are implemented through existing HTTP methods. The operations are: create, read, update, delete, patch, query, and action. The server MAY conditionally implement any operation.

5.1. Create

The "create" operation allows a client to create a new resource in a collection. A resource is created in one of two ways: if the client is requesting a specific identifier, then the resource is created via the HTTP PUT method; if the server is to select its own identifier, then the resource is created via the POST method. The PUT method SHOULD be preferred over POST.

5.1.1. PUT Request

Using the PUT method, the identifier of the resource to create is specified in the Request-URI of the HTTP request. The server MAY override the requested identifier of the resource and select a suitable identifier of its own.

To unambiguously request resource creation, the "If-None-Match" header MUST contain the value "*". If no precondition header unambiguously requests resource creation or update, the server MAY employ its own means of determining how to interpret the PUT method.

```
PUT /{collection}/{id} HTTP/1.1
Content-Type: application/json
If-None-Match: *
...

[JSON representation of resource to create]
```

5.1.2. POST Request

Using the POST method, the identifier of the resource is not specified. The Request-URI of the request MUST NOT contain a query component, to distinguish it from an action operation. The server MUST select a suitable identifier for the created resource.

```
POST /{collection} HTTP/1.1
Content-Type: application/json
...

[Resource representation]
```

5.1.3. Response

```
HTTP/1.1 201 Created
Content-Type: application/json
Location: [location of resource]
ETag: "[resource version]"
...

[Resource metadata object]
```

The "Location" header field contains a URI that identifies the newly created resource. The optional "ETag" header field contains the version of the newly created resource. The resource metadata object also contains the identifier and optional version of the newly created resource.

If version control is implemented for a given resource, the server MUST expose the new resource version in the ETag header of the response.

5.2. Read

The "read" operation allows a client to read a representation of a resource from the server. It is implemented using the HTTP GET method. The client MUST NOT include a query component in the Request-URI, to distinguish it from a query operation.

If the resulting representation of the resource is a JSON object, it

SHOULD contain the JSON "_id" member, and also the "_rev" member if resource version is supported by the server implementation.

5.2.1. Request

```
GET /{collection}/{id} HTTP/1.1
...
```

5.2.2. Response

```
HTTP/1.1 200 OK
Content-Type: application/json
ETag: "[resource version]"
...

[Resource representation]
```

If version control is implemented for a given resource, the server MUST expose the resource version in the ETag header of the response.

5.3. Update

The "update" operation allows a client to update the representation of a resource on the server. It is performed using the HTTP PUT method. To cause the PUT method to unambiguously request a resource update, the "If-Match" header MUST contain the current version of the resource. If no precondition header unambiguously requests resource creation or update, the server MAY employ its own means of determining how to interpret the PUT method.

5.3.1. Request

```
PUT /{collection}/{id} HTTP/1.1
Content-Type: application/json
If-Match: "[resource version]"
...

[Resource representation]
```

5.3.2. Response

```
HTTP/1.1 200 OK
Content-Type: application/json
ETag: "[resource version]"
...

[Resource metadata object]
```

If version control is implemented for a given resource, the server MUST expose the updated resource version in the ETag header of the response.

5.4. Delete

The "delete" operation allows a client to delete a resource, or optionally an all resources within a collection. It is performed using the HTTP DELETE method.

To delete a single resource, both the collection location and resource identifier MUST be specified in the Request-URI of the HTTP request. To delete all the resources within a collection, the collection location MUST be specified.

If deleting all resources within a collection is not supported by the server implementation, the server SHOULD respond to such requests with a 403 Forbidden error status code.

5.4.1. Request

```
DELETE /{collection}/{id} HTTP/1.1
If-Match: "[resource version]"
...
```

5.4.2. Response

```
HTTP/1.1 204 No Content
...
```

5.5. Patch

The "patch" operation allows a client to apply a set of partial modifications to a resource on the server. This is particularly useful if the client does not have permission to modify resources in their entirety.

The "patch" operation is performed using the HTTP PATCH method, per

[RFC5789]. The supported patch document format(s) to apply the partial modifications are determined by the server implementation.

5.5.1. Request

```
PATCH /{collection}/{id} HTTP/1.1
If-Match: "[resource version]"
...

[Patch document content]
```

5.5.2. Response

```
HTTP/1.1 200 OK
Content-Type: application/json
ETag: "[resource version]"
...

[Resource metadata object]
```

If version control is implemented for a given resource, the server MUST expose the updated resource version in the ETag header of the response.

5.6. Query

The "query" operation performs a parametric query of a resource or collection, and responds with a corresponding result. The execution of a query MUST NOT incur side effects. It is implemented using the HTTP GET method. The client MUST include a query component in the Request-URI to distinguish it from a read operation.

To query a single resource, both the collection location and resource identifier MUST be specified in the Request-URI of the HTTP request. To query an entire collection, the collection location MUST be specified.

5.6.1. Request

```
GET /{collection}/{id}?{query} HTTP/1.1
...
```

5.6.2. Response

```
HTTP/1.1 200 OK
Content-Type: application/json
...

[Query result value]
```

The structure of the query result value is determined by the server implementation.

5.7. Action

The "action" operation performs a parametric action on a resource or collection, and responds with an optional result. The execution of an action MAY incur side effects.

The operation is implemented via the HTTP POST request. The Request-URI MUST contain a query component, to distinguish it from a create operation. The request MAY include an entity body.

To perform an action on a single resource, both the collection location and resource identifier MUST be specified in the Request-URI. To perform an action on an entire collection, the collection location MUST be specified.

If the response contains a result value, then the server SHOULD respond with a 200 OK status code; otherwise it SHOULD respond with a 204 No Content status code.

5.7.1. Request (with entity body)

```
POST /{component}/{id}?{query} HTTP/1.1
Content-Type: [entity body content type]
...

[Entity body]
```

5.7.2. Request (without entity body)

```
POST /{component}/{id}?{query} HTTP/1.1
...
```

5.7.3. Response

```
HTTP/1.1 200 OK
Content-Type: application/json
...

[JSON action result]
```

If the response contains a result, then the server SHOULD respond with a 200 OK status code; otherwise it SHOULD respond with a 204 No Content status code.

The structure of the action response value (if any) is determined by the server implementation. If version control is implemented for a resource the action is being performed on, the server MAY expose the updated resource version in the ETag header of the response.

6. Request Context

A server MAY require some form of request context be established by the client prior to allowing access to resources. How such context is established, persisted and transmitted is out of the scope of this convention, and SHOULD be specified by the server implementation.

If inadequate request context has been established, the server SHOULD indicate this with a 401 Unauthorized error status code, unless there is another means of indicating this condition which is consistent with the required request context.

7. Access Control

The server implementation MAY enforce access control policies that restrict what resources a client can access and/or on what JSON values within each resource may be accessed.

If a sufficient request context has been established, but such context does not permit the requested access to a resource, the server SHOULD reject such requests with a 403 Forbidden error status code and some detail in a response error object describing the nature of the rejection.

The server implementation MAY amend representations of resources to conform to access control policies, and SHOULD specify under what conditions such amendments are applied.

8. Resource Validation

The server MAY enforce validation rules on resource representations provided by the client. If such a validation fails, the server SHOULD indicate this with a 403 Forbidden error status code and some detail in a response error object describing the nature of the validation failure.

9. Resource Metadata

Most responses to requests contain metadata about the resource being accessed. The metadata is included an HTTP ETag response header as well as members within a JSON object resource representation, including a JSON object specifically intended to contain only metadata (referred to within this document as a "resource metadata object").

9.1. Header

ETag
The current version of a resource, if version control is implemented for the resource.

9.2. Object Members

"_id": string, required
The identifier of the resource, relative to the collection it is a member of.

"_rev": string, optional
The current version of the resource, if version control is implemented for the resource.

10. Error Response

In the event of an error, a 4xx or 5xx HTTP status code SHOULD BE included in the response, with an entity body containing a JSON object that has minimally an "error" member.

The server implementation MAY provide additional members in the error object, which provide additional context and description of the nature of the error. Any such additional members SHOULD be specified by the server implementation.

```
   {
       "error": string,
       ...
   }
```

10.1. Members

"error": string, required
A mnemonic error code that expresses the type of error that occurred. The error code values and their associated meanings SHOULD be specified by the server implementation.

11. Modifying a Resource Identifier

The server MAY allow the update and/or patch operations to modify the identifier of a resource within the collection if the resource has a JSON object representation. If such modification is disallowed, the server SHOULD respond with a 403 Forbidden status code.

To indicate a request to modify the resource identifier, the "_id" metadata member should be included in the request entity and differ from the existing resource identifier in the Request-URI.

If the server successfully modifies the resource identifier, instead of responding with a 200 OK status code, the server MUST respond with a 201 Created status code, with a Location header containing the URI of the newly created resource.

If there is already a resource with the requested identifier, the server MUST respond with a 409 Conflict status code indicating it could not be modified. If the server rejects the identifier as invalid, the server SHOULD respond with a 403 Forbidden status code.

12. IANA Considerations

This document has no IANA actions.

13. Security Considerations

TBD.

14. Acknowledgements

The following individuals contributed ideas, feedback and wording to

this specification:

Alin Brici, Andi Egloff, Kornel Lesinski, Eve Maler, Ryder Ross, David Zarlengo, David Zuelke.

This convention was influenced by various projects that expose HTTP-based data access interfaces, especially those that managed JSON-based representations, notably CouchDB.

15. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[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.

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.

[RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, March 2010.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, March 2012.

[RFC6585] Nottingham, M. and R. Fielding, "Additional HTTP Status Codes", RFC 6585, April 2012.

Appendix A. Questions and Answers

Why this convention?

HTTP-based APIs seem to be repeatedly following similar patterns, but with enough differences to preclude common programmatic implementations. The intent of this convention is to provide a basic set of rules upon which a more application-specific interface can be specified (principle: DRY).

Is this a RESTful interface?

This convention strays from a pure REST interface, as it prescribes a specific (JSON) representation for resources, and arguably fails to adhere to the principle of hypermedia as the engine of application state (HATEOS).

What kind of interface is it?

It may be better classified as a resource-oriented interface. It establishes a uniform interface (set of operations), and maps between those operations and the standard methods provided by HTTP.

Why are ETags coupled to resource version, not entity value?

Entity tags in HTTP are intended to be used to compare two or more entities for the same resource. Since this convention establishes an exclusive representation of the resource (JSON), entity tag should be safely associable with resource version. RFC 2616 provides support for this practice in section 14.24 by stating, "It is also used, on updating requests, to prevent inadvertent modification of the wrong version of a resource."

Why are "query" and "action" operations abstract?

Queries and actions are too domain-specific to allow any more specificity than is expressed in this convention. Therefore, this convention does not attempt to define query/action semantics beyond the fact that they are parametric.

Appendix B. Examples

TBD.

Author's Address

Paul C. Bryan (editor)
Salesforce.com

Phone: +1 604 783 1481
Email: pbryan@anode.ca