Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: November 1, 2014

J. Fox, Ed.
ciferproject.org
J. Fox
University of Washington
K. Hazelton
University of Wisconsin
C. Hyzer
University of Pennsylvania
B. Oshrin
Internet2
B. Savage
Boston College
B. Thompson
Unicon
J. Vuccolo
Penn State
April 30, 2014

CIFER API Framework 01

ciferproject-org-framework-api-v1-draft-02

Abstract

The CIFER API Framework gives a consistant look, feel, syntax and nomenclature to all of the various CIFER resource representations.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on November 1, 2014.

Copyright Notice

Table of Contents

1. Introduction

The [CIFER] initiative includes work to define a set of shared APIs for core functions in identity and access management. One desired feature of such systems is a consistancy tructure and vocabulary of API URLs and representations.

2. Definitions

RESTful

This framework references Representational State Transfer (REST), where a service is composed of an addressable set of resources. URIs are used to address resources and transitions of a resources state are accomplished by direct, stateless client interactions with these URIs.

Clients of the service may:

- Search for resources,
- Get a representation of a resource,
- Add or replace a resource, and
- Delete a resource.

3. Requirements Language

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].

4. Authentication and Authorization

The framework does not specify authentication or authentication methods. Some acceptable methods include:

1. Basic auth over https
2. Client certificate auth over https
3. Encrypted and signed content over any transport

5. API Conventions

5.1. Grammar and Format Constraints

5.1.1. URI syntax

A major version of a particular CIFER API must be indicated in the URI path (for example, "/v2/"). A major version will differentiate incompatible representations, parameters or headers. A major version change is also indicated by a change in field type. Any minor version will be indicated in the response metadata not in the URI path (see response section below).

5.1.2. Request/response grammar

1. Default representation language is Javascript Object Notation (JSON).
2. XML can be requested, but support is not required.

5.1.3. Data and field formats

1. Character set is Unicode, encoded as UTF-8.
2. imestamp representations will comply with ISO 8601. For example: 2012-10-04T03:10:14.123Z
3. Durations also comply with ISO 8601, For example: P1.81S
4. Field names are camelCase (including acronyms), e.g. "selfUri"
5. Where there is discretion enum and other fixed field values should be camelCase.
6. Field names must start with a letter. They can contain upper/lower letters, numbers and underscores.
7. Field types must stay consistent within a major version.
8. Objects are identified by uri
    - For example: netId:mchyzer, or userId:12345678

5.2. Requests

5.2.1. Resource references

CIFER APIs accommodate large responses by recognizing full and partial resources.

1. A full resource, where the response is the full resource requested, must be identified by a URI only, not by query string or fragment.
2. A partial resource, like a response that is better served across pages, is identified by a URI that contains at least one additional query string parameter (see paging options in query string options section below).

5.2.2. Request methods

1. The request method of POST overridden by a header (see header options below).
2. A PUT request need not specify all attributes.
    1. Replacing a resource will not affect attributes not specified in the request.
    2. A necessary attribute not specified will return an error 400 (see response codes below).

3. Searches return resource URIs and default resource attributes.
    - Each service will document the fields returns on searches.
    - Searches may request "&extraFields=field3,field4,..&" to get those fields in addition to the default set.
    - Searches may request "&omitFields=field3,field4,..&" to omit those fields from the default set.
    - Support for these features is optional.

4. Searches may specify paging options by QS parameters
    - pageLimit: Number of entries per page
    - pageOffset: page number, first page is zero

5.2.3. Header Options

1. The request method of POST may be overridden by the header:
    - X-HTTP-Method-Override: GET | PUT | DELETE

2. XML representation may requested by header:
    - Accept: text/xml

3. Use of ETag and If-Match:
    - A request to replace a resource must specify an If-Match header.
        - The value must match the resource's current ETag value, or
        - The value may be a 'match-any' wildcard, '*'
        - Acceptance of wildcard is service dependent.
        - A request to add a resource must not specify an If-Match header.

4. A client may request to "act as" another user via a header:
    - X-CIFER-Act-As: user_uri

5.2.4. Query string options

1. Requests may specify field filters on returned resources.
    - Query string may specify a list desired fields: "&fields=field1,field2,...&"
    - A requests for an unknown field is not an error.
    - Support for this feature is optional.

2. Requests for large resources, e.g. group membership, may specify paging parameters
    - sortField:
    - sortOrder: 'a*', 'd*', case insensitive
    - pageLimit: number of entries per page, zero means no paging
    - Mutually exclusive page start:
        - pageOffset: page number, starting with zero, OR
        - pageStartValue: first value returned will be the one AFTER this value

    - The service may provide default paging parameters, and may reduce the requested limits.

3. Boolean parameters are generously interpreted:
    - true|false, or t|f or yes|no, or y|n or 1|0, case insensitive

4. A client may request a pretty-printed response with the query string, "indent=true".
5. Unexpected Query String parameters will be ignored.

5.3. Request methods

While most browser facing web sites are nowadays discoverable from the root resource, there never was a time when an application could be completely discovered through its API.

1. The complexity of resources, authorizations, capabilities, and interactions requires, often extensive, external documentation of the application.
2. To be efficient a client often must be able to directly address a resource.

These considerations indicate that a root resource, with links to version resources, with links to other resource URLs will in almost all cases be bypassed by clients. User readable API documentation will specify direct, if relative, links directly to most of the individual resources.

The framework does not specify these root and directory representations, except that they follow the general request and response guidelines. A root page may be used to provide general information about the application instance: what version it supports; whether or not it supports XML representation; etc.

5.4. Responses

5.4.1. Response structure

Resources are returned along with metadata about the resource and metadata about the response (see JSON examples that follow)

1. Resources are wrapped in an object which contains:
    1. resource: The actual resource for the request.
    2. resourceMetadata: metadata about the resource. fields as appropriate to the resource
        - lastModified: timestamp when the resource was modified
        - selfUri: URI to the current resource.
        - ETag, or equivalent, for resources that can be replaced, ...
    3. responseMetadata: metadata about this particular response
        - responseTimestamp:
        - responseTime: time that the server took in processing this request
        - requestCommentary: freeform text about the request that the server processed (for debug)
        - httpStatusCode: response status as an HTTP status code.
        - serverVersion: major.minor server version number
        - paging parameters
            - pageLimit
            - pageOffest
            - sortField
            - pageOffsetValue
            - sortOrder

5.4.2. Response codes

1. 200: the resource was found
2. 201: the resource was created
3. 400: the request was not valid
4. 401: the client was unauthenticated
5. 403: an authenticated client was not authorized for the request
6. 404: a requested resource was not found
7. 409: (conflict)
    - PUT to an existing resource without an If-Match
8. 412: (precondition failed)
    - If-Match header did not match resource's ETag, or
    - If-Match header supplied but resource not found (add with an if-match), or

    - wildcard if-match not allowed

9. 500: server error

6. Examples

6.1. Site directory

What might be returned by a top-level GET to a service.

```
{
  "resource":{
    "people": "/people",
    "places": "/places"
  },
  "resourceMetadata":{
    "lastModified":"2012-11-04T09:57:03.541Z",
    "baseUri":"https://example.u.edu/ppservice/v1/",

  },
  "responseMeta":{
    "responseTimestamp":"2013-07-04T09:57:03.541Z",
    "httpStatusCode":200,
    "responseTime":"P0.011S"
    "serverVersion":"1.1"
  }
}
```

6.2. Search response

What might be returned by a search GET to a service.

```
{
  "resource":[
   {"uuid": "3ac85497-1663-4ef4-9933-3df8bc7800bc",
    "lastName": "Doe",
    "givenName": "John A.",
    "address": { ... }
    "status": "something",
    "uri": "/v1/people/3ac85497-1663-4ef4-9933-3df8bc7800bc"
   },
   {"uuid": "3ac85497-1663-4ef4-9933-3df8bc7800cc",
    "lastName": "Doe",
    "givenName": "John B.",
    "address": { ... }
    "status": "something",
    "uri": "/v1/people/3ac85497-1663-4ef4-9933-3df8bc7800cc"
   },
  ],

  "resourceMetadata":{
    "baseUri":"https://example.u.edu/ppservice",
    "sortOrder":"a",
    "pageLimit":100,
    "pageOffset":0,
    "sortField":"name",
    "statusCode":"SUCCESS",
  },
  "responseMeta":{
    "responseTimestamp":"2013-07-04T09:57:03.541Z",
    "httpStatusCode":200,
    "responseTime":"P0.011S"
    "serverVersion":"1.1"
  }
}
```

7. IANA Considerations

This memo includes no request to IANA.

8. Security Considerations

The API framework does not specify how services identify their clients.

9. Acknowledgements

This document was produced by the CIFER Shared API work team. People who made particular contributions to the work include Jim Fox, Keith Hazelton, Chris Hyzer, Benn Oshrin, Jonathan Pass and Jimmy Vuccolo

10. References

10.1. Normative References

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

10.2. Informative References

[CIFER] CIFER, "CIFER Home Page", 2014.

Authors' Addresses

Jim Fox (editor)
ciferproject.org
4333 Brooklyn Ave NE
Seattle, Washington 98195
US
Phone: +1 206 543 4320
EMail: fox@uw.edu

Jim Fox
University of Washington
EMail: fox@uw.edu

Keith Hazelton
University of Wisconsin
EMail: hazelton@wisc.edu

Chris Hyzer
University of Pennsylvania
EMail: mchyzer@isc.upenn.edu

Benn Oshrin
Internet2
EMail: benno@internet2.edu

Brian Savage
Boston College
EMail: brian.savage@bc.edu

Bill Thompson
Unicon
EMail: wgthom@gmail.com

Jimmy Vuccolo
Penn State
EMail: jvuccolo@psu.edu