CIFER API Framework

As of Q3 2015, the CIFER Project has been transitioned to the TIER-Data Structures and APIs Working Group.

About the CIFER API Framework

These are uniform API conventions adopted by teams working across CIFER APIs. It provides a reference for current and future contributors to the set of CIFER APIs.

REST as Reference Style for CIFER API Framework

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

Clients of the service may:

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

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. Oauth Bearer tokens over https
4. Encrypted and signed content over any transport

API

Grammar and Format Constraints

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).

Request/response grammar

Javascript Object Notation (JSON) is standard for request and responses.
XML may be used as an additional response format but is not required.

Data and field formats

1. UTF-8 is the character set encoding.
2. Timestamp representations will comply with ISO 8601. For example, “2014-03-04T23:20:28Z” represents a time and “PT1.81S” represents a time duration.
3. Attribute names are camelCase (including acronyms). For example, "selfUri".
4. Enum and other fixed attribute values should be camelCase.
5. Attribute names must conform to the following ABNF [RFC5234] rules:
    a. ATTRNAME = ALPHA *(nameChar)
    b. nameChar = "-" / "_" / DIGIT / ALPHA
6. Attribute types must stay consistent within a major version.
7. Objects are identified by uri: e.g.: netId:mchyzer, or userId:12345678
8. Where a hierarchy is required within a path element, the colon (“:”) should be used as a separator.

Requests

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).
3. Because of their disconnected nature, paged resources may be incomplete or inconsistent.

Request methods

1. The request method of POST may be overridden by an HTTP header (see header options below)
2. A PUT request need not specify all attributes.
    a. Replacing a resource will not affect attributes not specified in the request.
    b. A necessary attribute not specified will return an error 400 (see response codes below).
3. Search requests return resource URIs and default resource attributes.
4. Each service must document the default fields returned on each search request type.
5. Search requests may use a number of query string options to accomplish paging operation, response field selection and other search request extensions (see query string options section below).

HTTP header options

1. The request method of POST may be overridden by the header:
    a. **X-HTTP-Method-Override: GET | PUT | DELETE**
2. XML representation of resources are not required, but may be requested by the header:
    a. **Accept: text/xml**
3. **ETag** and **If-Match** may be used given the following constraints:
    a. A request to add a resource must not specify an **If-Match** header.
    b. A request to replace a resource must specify an **If-Match** header.
    c. The **ETag** value provided by the client must either match the resource's current ETag value, or the ETag value must contain a “match any” wildcard (as with “*”, where acceptance of the wildcard is service dependent).
4. A client may request to "act as" another user via the header:
    a. **X-CIFER-Act-As:** user_uri
5. A client may define or override a request header with a query string parameter, e.g.:
    a. "**&Accept**=application%2Fjson"

Specifying Attributes

Requests may request specific attributes in the returned resources with the query string parameters:

1. *&attributes=*attribute1,attribute2,...
    a. Include only these attributes
2. *&extraAttributes=*attribute1,attribute2,...
    a. Include these in addition to the default
3. *&omitAttributes=*attribute1,attribute2,...
    a. Remove these default attributes from the response
4. A request for unknown attributea will not cause an error.
5. Support of this feature is optional.

Pagenation and sort order

Requests for large resources, like membership in a large group, may specify paging query string parameters :

- **sortBy**: name of attribute
- **sortOrder**: “a*” or “A” for ascending, or “'d*” or “D” for descending
- **count**: number of entries per page, zero means no paging
- **startIndex**: row number of the first resource,
- **startPage**: page number, starting with zero
- **startValue**: first value returned will be the one ***after*** this value
    - value is the value of the sortBy attribute

A service may provide default paging parameters, and may reduce the limits requested via query string parameters.

Misc

1. A client may request a pretty-printed response using:
    a. **&indent=true**
    b. The meaning o{*}f '*indent' is up to the service.
2. Boolean parameters should be generously interpreted:
    a. e.g.: true|false”, “t|f”, “yes|no”, “y|n”, “1|0”
3. Undefined query string parameters will not cause an error.

Root and application information resources

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.

Responses

Response structure

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

ciferMeta:

Individual ciferMeta attributes are optional.

1. **lastModified**: timestamp when the resource was modified
2. **selfUri**: URI to the current resource
3. **ETag**, or equivalent, for resources that can be replaced
4. **responseTimestamp:** when the response was issued
5. **responseTime**: time that the server took in processing this request
6. **requestCommentary**: freeform text about the request that the server processed (for debug)
7. **httpStatusCode**: response status as an HTTP status code.
8. **serverVersion**: major.minor server version number
9. **pagenation**: (echo of requested pagenation and sort parameters)

Response codes

- 200: the resource was found
- 201: the resource was created
- 400: the request was not valid
- 401: the client was unauthenticated
- 403: an authenticated client was not authorized for the request
- 404: requested resource was not found
- 406: service cannot provide requested XML representation
- 409: (conflict)
    - PUT to an existing resource without an If-Match
- 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
- 500: server error

JSON syntax examples

example site directory

```
{
  "people": "/people",
  "places": "/places",
  "ciferMeta":{
    "lastModified":"2012-11-04T09:57:03.541Z",
    "baseUri":"https://example.u.edu/ppservice/v1/",
    "responseTimestamp":"2013-07-04T09:57:03.541Z",
    "httpStatusCode":200,
    "responseTime":"P0.011S",
    "serverVersion":"1.1"
  }
}
```

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