Citizens Agenda Documentation

Release 0.1

Oxys SARL

Sep 27, 2017

Contents

Contents:

CHAPTER 1

Start with oauth and account registration

curl-oauth

curl-oauth is a curl like tool with oauth support. Sample below use it. You can download it at github

Registration

To access api you must have an account and oauth tokens used to sign requests to api. So go to this url and proceed with account registration.

Second step is to create a consumer that will be allowed to access the api, fill the form here and validate. You have now the token and the secret for your new consumer.

Third step is to get access token and access secret for this consumer. Achieve this by using curl-oauth:

```
>>> export API="http://api.devel.czagenda.org/api;
        curl-oauth --domain cz-api \
          --request-token \
          --consumer-secret <CONSUMER_SECRET> \
          --consumer-key <CONSUMER_TOKEN>  \
          --request-token-url $API/oauth-token/_request-token \
          --request-token-callback http://www.oxys.net \
          --request-token-authorize-url http://auth.czagenda.oxys.net/oauth/
↪authorize/
          --request-token-access-token-url $API/oauth-token/_access-token
```

Follow instructions, the pin asked is the oauth_verifier parameter provided in querystring of the callback url.

At the end of process you can request api using curl-oauth –domain cz-api.

CHAPTER 2

RESTful API

Overview of the RESTful API

How it works

Each of the api's URI match a document or a subset of documents. HTTP methods define actions to do on the document(s).

Theses methods are:

- GET : retrieve a document or a subset of documents
- POST : create a new document
- PUT : update a document
- DELETE : delete a document

Data format

Only JSON format is supported. You must set the Content-Type header at application/json in all requests to the server.

API responses

The API's responses contain the standards HTTP codes which are significant of what happened.

- 200 OK : all went successfully gone
- 201 Created : a new document was created
- 204 Deleted : a document was deleted
- 400 Bad Request : the request is malformed
- 401 Forbidden : document ressource is forbidden

- 404 Not Found : document is not found
- 500 Internal : Server Error : a server internal error occurred

API structure

URIs

URI are prefixed with /api/, followed by the document's type and optionnaly by the document identifier.

URIs examples

```
>>> /api/<DOCUMENT_TYPE>/
```

```
>>> /api/<DOCUMENT_TYPE>/<DOCUMENT_ID>
```

Playing with the API

curl-oauth is used in examples bellow to make request to the server. If you're not familiar with please take a look at https://github.com/oxys/curl-oauth

1. Get a document's list

   It's done by invoking the GET HTTP method on an URI like /api/<DOCUMENT_TYPE>/

   Request

   ```
   >>> curl-oauth --domain cz-api -X GET http://api.devel.czagenda.org/api/
   ↪<DOCUMENT_TYPE>/
   ```

   Response

   The server returns data as below

   ```
   >>> {
       "total_rows":<Database Document's count>,
       "offset": 0,
       "rows":[{
               Document data...
               }]
       }
   ```

   - total_rows is the document count of the specified type in the database, not in the response.
   - rows contains the documents, see data structure for more explanations.
   - offset is the amount of documents that are skipped when request is paginated.

2. Get document's count

   To get only the amount of documents, use

   ```
   >>> curl-oauth --domain cz-api -X GET http://api.devel.czagenda.org/api/
   ↪<DOCUMENT_TYPE>/_count
   ```

   Methods allowed are GET and POST. Search query syntax can also be applied on these urls.

3. Get a document

It's done by invoking the GET HTTP method on an URI like /api/<DOCUMENT_TYPE>/<DOCUMENT_ID>

Request

```
>>> curl-oauth --domain cz-api -X GET http://api.devel.czagenda.org/api/
↪<DOCUMENT_TYPE>/<DOCUMENT_ID>/
```

Response

The server returns one document

```
>>> { Document data... }
```

4. Create a document

   It's done by invoking the POST HTTP method on an URI like /api/<DOCUMENT_TYPE>/

   Request

```
>>> curl-oauth --domain cz-api --json -X POST http://api.devel.czagenda.
↪org/api/<DOCUMENT_TYPE>/  -d '{
                "an_attribute" : "a value", "another_attribute" :
↪"another value"
        }'
```

   Response

   The server returns the newly created document with some system attributes such as _id or _rev. Systems attributes are described HERE

5. Update a document

   It's done by invoking the PUT HTTP method on an URI like /api/<DOCUMENT_TYPE>/<DOCUMENT_ID>

   Request

```
>>> curl-oauth --domain cz-api --json -X PUT http://api.devel.czagenda.
↪org/api/<DOCUMENT_TYPE>/  -d '{
                "an_attribute" : "an updated value", "another_attribute"␣
↪: "another value"
        }'
```

   Response

   The server returns the updated document.

6. Delete a document

   It's done by invoking the DEL HTTP method on an URI like /api/<DOCUMENT_TYPE>/<DOCUMENT_ID>

   Request

```
>>> curl-oauth --domain cz-api -X DELETE http://api.devel.czagenda.org/
↪api/<DOCUMENT_TYPE>/<DOCUMENT_ID>
```

   Response

   The server returns nothing

Specifics of the API by document type

User

User base uri is /user

1. Create

```
>>> curl-oauth --domain cz-api --json -X POST http://api.devel.czagenda.
↪org/user/  -d '{
                        "firstName" : "John",
                        "lastName" : "Doe",
                        "email" : "john.doe@domain.com",
                        "login" : "john.doe"
                }'
```

Server response with the 201 code

```
>>>     {
          "id": "/user/johndoe",
          "createDate": "2011-10-04T08:19:16.753Z",
          "updateDate": "2011-10-04T08:19:16.753Z",
          "login": "john.doe",
          "firstName": "John",
          "lastName": "Doe",
          "isActive": false,
          "isStaff": false,
          "isSuperuser": false,
          "lastSeen": null,
          "dateJoined": null,
          "groups": "/user/johndoe/groups"
        }
```

Note that id was generated from the provided login.

2. Update

You can make a request with partial data or with a complete data structure.

```
>>> curl-oauth --domain cz-api --json -X PUT http://api.devel.czagenda.
↪org/user/johndoe?pretty=true  -d '{
                "firstName" : "Jack"
        }'
```

Server response with the 200 code

```
>>> {
          "id": "/user/johndoe",
          "createDate": "2011-10-04T08:19:16.753Z",
          "updateDate": "2011-10-04T08:24:30.840Z",
          "login": "john.doe",
          "firstName": "Jack",
          "lastName": "Doe",
          "isActive": false,
          "isStaff": false,
          "isSuperuser": false,
          "lastSeen": null,
          "dateJoined": null,
          "groups": "/user/johndoe/groups"
        }
```

2. Delete

```
>>> curl-oauth --domain cz-api --X DELETE http://api.devel.czagenda.org/
↪user/johndoe
```

Server response with the 204 code and an empty body.

Group

Group base uri is /group

1. Create

```
>>> curl-oauth --domain cz-api --json -X POST http://api.devel.czagenda.
↪org/group/?pretty=true  -d '{
                "title" : "My group",
                "description" : "Description of my first group"
        }'
```

Server response with the 201 code

```
>>> {
          "id": "/group/my-group",
          "createDate": "2011-10-04T08:32:01.231Z",
          "updateDate": "2011-10-04T08:32:01.231Z",
          "title": "My group",
          "description": "Description of my first group",
          "writeGroups": "/group/my-group/perms/wg",
          "writeUsers": "/group/my-group/perms/wu",
          "users": "/group/my-group/users"
        }
```

Note that id was generated from the provided title.

Membership

Membership base uri is /membership

1. Create

```
>>> curl-oauth --domain cz-api --json -X POST http://api.devel.czagenda.
↪org/membership/  -d '{
                        "user" :  "/user/johndoe",
                        "group" : "/group/my-group"
                }'
```

Server response with the 201 code

```
>>> {
          "id": "/membership/0b3bcb3a4b4fd153e2373f7ec49f5a57",
          "createDate": "2011-10-04T08:40:32.117Z",
          "updateDate": "2011-10-04T08:40:32.117Z",
          "group": "/group/my-group",
          "user": "/user/johndoe"
        }
```

A membership document is a relation between a user and a group.

2. Get user's groups

   If you want to list all groups for a user just query the uri stored in the "group" attribute of the user document

   ```
   >>> curl-oauth --domain cz-api -X GET http://api.devel.czagenda.org/user/
   ↪johndoe/groups/
   ```

   Server response

   ```
   >>> {
             "total_rows": 1,
             "offset": 0,
             "rows": [
               {
                 "group": "/group/my-group",
                 "createDate": "2011-10-04T08:40:32.117Z",
                 "updateDate": "2011-10-04T08:40:32.117Z",
                 "id": "/membership/0b3bcb3a4b4fd153e2373f7ec49f5a57"
               }
             ]
           }
   ```

   The server will return all membership documents for the user. Observe that user attribute of the membership document is not here.

   If you want in the same query fetch the group document you can do it with the special query string parameter include_docs

   ```
   >>> curl-oauth --domain cz-api -X GET http://api.devel.czagenda.org/user/
   ↪johndoe/groups/?include_docs=true
   ```

   server response

   ```
   >>> {
             "total_rows": 1,
             "offset": 0,
             "rows": [
               {
                 "group": {
                   "title": "My group",
                   "users": "/group/my-group/users",
                   "description": "Description of my first group",
                   "writeUsers": "/group/my-group/perms/wu",
                   "createDate": "2011-10-04T08:32:01.231Z",
                   "updateDate": "2011-10-04T08:32:01.231Z",
                   "writeGroups": "/group/my-group/perms/wg",
                   "id": "/group/my-group"
                 },
                 "createDate": "2011-10-04T08:40:32.117Z",
                 "updateDate": "2011-10-04T08:40:32.117Z",
                 "id": "/membership/0b3bcb3a4b4fd153e2373f7ec49f5a57"
               }
   ```

```
            ]
        }
```

3. Get group's users

   You can fetch the group's members by requesting on the "users" attribute of the group document

   ```
   >>> curl-oauth --domain cz-api -X GET http://api.devel.czagenda.org/
   ↪groups/my-group/users/
   ```

   or

   ```
   >>> curl-oauth --domain cz-api -X GET http://api.devel.czagenda.org/
   ↪groups/my-group/users/?include_docs=true
   ```

4. Update

   Warning: Update are not allowed on membership uri.

5. Delete

   To delete a membership relation, proceed in the same way as others documents

   ```
   >>> curl-oauth --domain cz-api --X DELETE http://api.devel.czagenda.org/
   ↪membership/0b3bcb3a4b4fd153e2373f7ec49f5a57
   ```

Agenda

Agenda base uri is /agenda

1. Create

   ```
   >>> curl-oauth --domain cz-api --json -X POST http://api.devel.czagenda.
   ↪org/agenda/  -d '{
                           "title" : "My private agenda",
                           "description" : "description of my private agenda"
                   }'
   ```

   Server response

   ```
   >>> {
     "id": "/agenda/my-private-agenda",
     "createDate": "2011-10-04T09:06:38.071Z",
     "updateDate": "2011-10-04T09:06:38.071Z",
     "title": "My private agenda",
     "description": "description of my private agenda",
     "writeGroups": "/agenda/my-private-agenda/perms/wg",
     "writeUsers": "/agenda/my-private-agenda/perms/wu"
   }
   ```

   Note that id was generated from the provided title.

   Warning: Agenda is used on event to restrict creation, update and deletion. To do an operation on an event that is part of an agenda you must have write access on the agenda.

Category

Category base uri is /category

Warning: Create, update or delete a category require staff privilege.

1. Create

```
>>> curl-oauth --domain cz-api --json -X POST http://api.devel.czagenda.
↪org/category/  -d '{
                        "title" : "A category",
                        "description" : "a description"
                }'
```

Server response

```
>>> {
  "id": "/category/b31398e4e0de03ef76bb168e32e41948",
  "createDate": "2011-10-04T09:06:38.071Z",
  "updateDate": "2011-10-04T09:06:38.071Z",
  "title": "A category",
  "description": "a description"
}
```

Event

Event base uri is /event

1. Create

```
>>> curl-oauth --domain cz-api --json -X POST http://api.devel.czagenda.
↪org/api/event  -d '{
                "event" : {
                        "title" : "My first event",
                        "where" : [{"valueString" : "Somewhere on earth␣
↪planet !"}],
                        "links" : [{"rel" : "describedby", "href" : "/
↪schema/event"}],
                        "category" : "/category/
↪b31398e4e0de03ef76bb168e32e41948"
                }
        }'
```

Data structure in the event attribute is written according to schema /schema/event. see schemas

Server response

```
>>> {
          "id": "/event/b31398e4e0de03ef76bb168e32e41948",
          "createDate": "2011-10-04T09:14:28.281Z",
          "updateDate": "2011-10-04T09:14:28.281Z",
          "event": {
            "title": "My first event",
            "where" : [{"valueString" : "Somewhere on earth planet !"}],
                "links" : [{"rel" : "describedby", "href" : "/schema/event
↪"}],
                "category" : "/category/b31398e4e0de03ef76bb168e32e41948"
          },
          "author": "/user/johndoe",
          "writeGroups": "/event/b31398e4e0de03ef76bb168e32e41948/perms/wg
↪",
          "readGroups": "/event/b31398e4e0de03ef76bb168e32e41948/perms/rg
↪",
          "writeUsers": "/event/b31398e4e0de03ef76bb168e32e41948/perms/wu
↪",
          "readUsers": "/event/b31398e4e0de03ef76bb168e32e41948/perms/ru",
          "agenda": null
        }
```

Note that author was automatically added according to your oauth domain

2. Moderate

   Events have moderate attributes which are approvedBy and disapprovedBy. These attributes are populate with special urls:

   - /api/event/<EVENT_ID>/moderate/approve
   - /api/event/<EVENT_ID>/moderate/disapprove

   These urls accept POST and DELETE http methods.

   Operation is that when a request is made on one of these urls. The attributes approvedBy and disapprovedBy are populated with the request user id.

   Approve an event

```
>>> curl-oauth --domain cz-api -X POST http://api.devel.czagenda.org/api/
↪event/b31398e4e0de03ef76bb168e32e41948/moderate/approve
```

   Disapprove an event

```
>>> curl-oauth --domain cz-api -X POST http://api.devel.czagenda.org/api/
↪event/b31398e4e0de03ef76bb168e32e41948/moderate/disapprove
```

   Remove an approval

```
>>> curl-oauth --domain cz-api -X DELETE http://api.devel.czagenda.org/
↪api/event/b31398e4e0de03ef76bb168e32e41948/moderate/approve
```

   Remove a disapproval

```
>>> curl-oauth --domain cz-api -X DELETE http://api.devel.czagenda.org/
↪api/event/b31398e4e0de03ef76bb168e32e41948/moderate/disapprove
```

   To fetch events that has been approved, a search request can be done like this. See *search* for more informations about searching.

```
>>> curl-oauth --domain cz-api  -X DELETE http://api.devel.czagenda.org/
↪api/event/_search -d 'q=approvedBy:/user/johndoe'
```

Entity

Entity base uri is /entity

1. Create

```
>>> curl-oauth --domain cz-api --json -X POST http://api.devel.czagenda.
↪org/api/entity  -d '{
                "entity" : {
                        "type" : "organization",
                        "name" : "My organization",
                        "where" : [{"valueString" : "Somewhere on earth␣
↪planet !"}],
                        "links" : [{"rel" : "describedby", "href" : "/
↪schema/organization"}]
                }
        }'
```

Data structure in the entity attribute is written according to schemas /schema/organization or /schema/person. see schemas

Server response

```
>>> {
          "writeUsers": "/entity/74319158a22cb48e4cf2b8aa32344695/perms/wu
↪",
          "author": "/user/johndoe",
          "createDate": "2011-11-02T08:27:44.115Z",
          "writeGroups": "/entity/74319158a22cb48e4cf2b8aa32344695/perms/
↪wg",
          "entity": {
            "where": [
              {
                "valueString": "Somewhere on earth planet !"
              }
            ],
            "type": "organization",
            "name": "My organization",
            "links": [
              {
                "href": "/schema/organization",
                "rel": "describedby"
              }
            ]
          },
          "updateDate": "2011-11-02T08:27:44.115Z",
          "id": "/entity/74319158a22cb48e4cf2b8aa32344695"
        }
```

Note that author was automatically added according to your oauth domain

Permission

Permission documents are used to define access rights on documents.

A permission document is composed of two attributes:

- grantTo which defines who has the permission. It could be a group or an user.
- applyOn which defines the document on which the permission is applied.

All documents types don't have all permissions types. More information can be found here

To create a permission on a document:

```
>>> curl-oauth --domain cz-api --json -X POST http://api.devel.czagenda.org/api/perms/
↪<DOCUMENT_TYPE>/<PERMISSION_CODE> -d '{
                "applyOn" : "<DOCUMENT_ID>",
                "grantTo" : "<GROUP_OR_USER_ID>"
        }'
```

Base permission uri varies depending on the document type and the permission type.

Permissions types and code are:

- write user : wu
- read user : ru
- write group : wg
- read group : rg

Special values for grantTo to grant privileges to all users or to all groups are /user/all and /group/all.

For example, to grant write privilege to group /group/my-group on an event document

```
>>> curl-oauth --domain cz-api --json -X POST http://api.devel.czagenda.org/api/perms/
↪event/wg -d '{
                "applyOn" : "/event/b31398e4e0de03ef76bb168e32e41948",
                "grantTo" : "/group/my-group"
        }'
```

Searches

Documents searches are restricted to types event, agenda, group, entity and user.

Searches urls are:

- http://api.devel.czagenda.org/api/event/_search
- http://api.devel.czagenda.org/api/agenda/_search
- http://api.devel.czagenda.org/api/group/_search
- http://api.devel.czagenda.org/api/entity/_search
- http://api.devel.czagenda.org/api/user/_search

Methods allowed are POST and GET.

Search query is contained in the q attribute:

```
>>> q=fulltext search field1:... field2:...>
```

Field syntax

To query a field that is not on top of the field hierarchy, use a single point to chain fields. For example, a document such as below

```
>>> {
        "field1" : {"field2" : "value"}
        }
```

can be queried with

```
>>> q=field1.field2:value
```

Text searches

This kind of search is applied on fulltext searches and text field searches. So searches examples below can be rewrite as:

```
>>> q=field:<search sample>
```

1. Phrase searches

   A Phrase is a group of words surrounded by double quotes such as "hello dolly".

   ```
   >>> q="event title"
   ```

2. Wildcards searches

   Single and multiple characters wildcard search are supported.

   For single character wildcard use : "?"

   ```
   >>> q=some?hing
   ```

   For multiple characters wildcard use : "*"

   ```
   >>> q=some*ng
   ```

   Wildcards can be placed at the beginning, end or in the middle of the search string.

   Wildcards are not useable with phrase search.

3. Fuzzy searches

   Fuzzy searches are based on the Levenshtein Distance. To do a fuzzy search use the tilde, "~", symbol at the end of a Single word Term.

   ```
   >>> q= somethink~
   ```

   This search will find terms like something.

   Fuzzy searches are not useable inside a phrase search.

4. Boolean operators

   Boolean operators allow terms to be combined through logic operators. Operators supported are and, "+", or, not and "-".

   You can use boolean operator to combine term and phrase query.

   The OR operator is the default conjuncton operator.

   ```
   >>> q=event title
   ```

   is exactly the same as

   ```
   >>> q=event or title
   ```

   The plus operator define a term as required.

```
>>> q=+event title
```

means that documents MUST contain event and MAY contain title.

The not operator excludes documents that contain the term after not.

To search document that contain “first stage” but not “second stage” use the query:

```
>>> q="first stage" not "second stage"
```

The not operator cannot be used with just one term. For example, the following search will return no results:

```
>>> q=not "second stage"
```

The “-” or prohibit operator excludes documents that contain the term after the “-” symbol.

```
>>> q=-"second stage"
```

5. Grouping

   Use parentheses to group clauses to form sub queries. This can be very useful if you want to control the boolean logic for a query.

```
>>> q=(second OR first) and stage
```

Date searches

Date are formated as:

```
>>> YYYY-MM-DD
```

Datetime are formated as:

```
>>> YYYY-MM-DDThh:mm:ssZ
```

1. Exact date searches

```
>>> q=dateField:2011-12-31
>>> q=dateField:2012-01-01
```

2. Date range searches

   Date range syntax is [from TO to]. The range operator “TO” must be CAPS.

   From and to can be a date, datetime, NOW or *.

   To search for documents that were created last year use this query:

```
>>> q=createDate:[2010-01-01 TO 2010-12-31]
```

   To search for documents that were created this year use this query:

```
>>> q=createDate:[2011-01-01 TO NOW]
```

   NOW will be replaced by current date

   To search for documents that were created before now use this query:

```
>>> q=createDate:[* TO NOW]
```

Boolean searches

Boolean values are TRUE or FALSE. They must be CAPS.

```
>>> q="isActive:TRUE"
```

Geographic searches

In bounding box and distance searches are supported.

1. In bounding box searches

   Use this query to search documents that take place in a bounding box

   ```
   >>> q=geoField:[TOPLEFT_LON TOPLEFT_LAT BOTTOM_RIGHT_LON BOTTOM_RIGHT_LAT]
   ```

2. Distance searches

   Use this query to search document that take place at 30 km from the point -0.3686 (lon), 43.3017 (lat)

   ```
   >>> q=geoField:[-0.3686 43.3017 DISTANCE 30km]
   ```

   Units supported for distance value are kilometers (km), or miles (mi)

Multiple searches for a same field

To apply multiple search query on a same field, just define the query more than one

```
>>> q=dateField:2011-12-03 dateField:2011-12-05
```

Special characters

The current list special characters are: + - && || ! ( ) { } [ ] ^ ” ~ * ? :

To escape these character use the before the character.

Pagination

Pagination is done by passing extra parameters in query string or body according to http method. These parameters are from and size.

from define the amount of documents to skip, size define the amount of documents to fetch.

For example

```
>>> curl-oauth --domain cz-api -X GET http://api.devel.czagenda.org/api/<DOCUMENT_
↪TYPE>/?from=10&size=5
```

This query will return documents from 10th to 15th

Defaults are size=10 and from=0.

Sorting

Sort is done by passing a "sort" parameter in query string or body according to http method.

Sort fields depends on document types.

For event:

- createDate
- updateDate
- event.title
- event.when.startTime
- event.when.endTime
- distance (only if a distance query was done)

For agenda and group:

- createDate
- updateDate
- title

For entity:

- createDate
- updateDate
- entity.name
- entity.firstName
- entity.lastName
- distance (only if a distance query was done)

For user:

- createDate
- updateDate
- lastSeen
- joinedDate
- login
- firstName
- lastName

For example, to sort events by distance ascendant and start date descendant, use the query below

```
>>> sort=distance -event.when.startTime
```

Sample usages

Query that retrieve events that took place in 2010 and contain "foundation", ordered by start date, paginated by 20 documents

```
>>> curl-oauth --domain cz-api -X GET 'http://api.devel.czagenda.org/api/event/?
↪from=0&size=20&q=foundation event.when.startTime:[2010-01-01 TO 2010-12-31]&
↪sort=event.when.startTime'
```

or

```
>>> curl-oauth --domain cz-api -X POST http://api.devel.czagenda.org/api/event/ -d
↪'from=0&size=20&q=foundation event.when.startTime:[2010-01-01 TO 2010-12-31]&
↪sort=event.when.startTime'
```

or

```
>>> curl-oauth --domain cz-api --json -X POST http://api.devel.czagenda.org/api/event/
↪ -d '{
                "from":0,
                "size":20,
                "q":"foundation event.when.startTime:[2010-01-01 TO 2010-12-31]",
                "sort":"event.when.startTime"
        }'
```

CHAPTER 3

Types of documents

Types of documents are:

- *event*
- *user*
- *group*
- *schema*
- *agenda*
- *category*

Common attributes

Warning: These attributes are read only.

`id (String)`

It is the unique identifier of the document. It is also its URI. So you can do a GET request on a document's id to get it.

`createDate (Datetime)`

creation date.

`updateDate (String)`

Last update date.

Type *user*

These documents holds informations about registered users.

`firstName (String)`

`lastName (String)`

`email (String)`

Warning: To be the user or to have a staff account is required to write/update this attribute

`password (String)`

Warning: To be the user or to have a staff account is required to write/update this attribute

`isActive (Boolean)`

Warning: Staff account is required to write/update this attribute

True if the user is active. Inactive users cant’s access the service.

`isStaff (Boolean)`

Warning: Staff account is required to write/update this attribute

True if the user is a staff member. Only staff members can access the admin interface.

`isSuperuser (Boolean)`

Warning: Staff account is required to write/update this attribute

True if the user is a super user. Super users have no restrictions when accessing the admin interface.

`lastSeen (Datetime)`

Warning: Read only

Date of last authentication against the service.

`dateJoined (Datetime)`

Warning: Read only

Date of the registration.

`groups (String)`

Warning: Read only

Contains an URI to get groups list of which the user is a member.

Type *event*

`event (Object)`

event attribute contains a data structure that describes the event itself. this structure is constrained by a *schema* document type.

`author (String)`

Warning: Read only

The user *id* of the *user* document type.

`writeGroups (List)`

Warning: Read only

Contains an URI to get write group permissions.

`readGroups (List)`

Warning: Read only

Contains an URI to get read group permissions.

`writeUsers (List)`

Warning: Read only

Contains an URI to get write user permissions.

`readUsers (List)`

Warning: Read only

Contains an URI to get read user permissions.

Type *schema*

These documents are used to describe and validate the *event* attribute for the *event* document type

`name (String)`

`schema (Object)`

Contains the data structure used to validate. More informations can be found here

`final (Boolean)`

Warning: Staff account is required to write/update this attribute

True if the document can be used to validate an event. If false the document must be part of an inheritance.

`sample (Object)`

Contains a data sample that validate the schema.

`template (String)`

Contains a template which can be used to render an event as html.

`status (Enum)`

Warning: Staff account is required to write/update this attribute

Define document's status

- PUBLISHED
- DRAFT
- DEPRECATED

Type *entity*

`entity (Object)`

entity attribute contains a data structure that describes the entity itself. this structure is constrained by a *schema* document type.

`author (String)`

Warning: read only

The user *id* of the *user* document type.

`writeGroups (List)`

Contains an URI to get write group permissions.

`writeUsers (List)`

Warning: read only

Contains an URI to get write user permissions.

Type *category*

Warning: Create, update and delete require staff privileges

Category are used in event.

`title (String)`

`description (String)`

`author (String)`

> **Warning:** read only

The user *id* of the *user* document type.

Type *group*

`title (String)`

`description (String)`

`users (String)`

> **Warning:** Read only

Contains an URI to get group memberships in terms of the group.

`writeGroups (List)`

> **Warning:** Read only

Contains an URI to get write group permissions.

`writeUsers (List)`

> **Warning:** Read only

Contains an URI to get write user permissions.

Type *agenda*

`title (String)`

`description (String)`

`writeGroups (List)`

Warning: Read only

Contains an URI to get write group permissions.

`writeUsers (List)`

Warning: Read only

Contains an URI to get write user permissions.

Type *membership*

`user (String)`

Contains the user id

`group (String)`

Contains the group id

CHAPTER 4

Schemas

Goal of schema is to describe data structure. see here for more details.

Documents that use schema to validate must include the links attribute. See base schema

```
>>> curl-oauth --domain cz-api -X GET http://api.devel.czagenda.org/api/schema/base-
↪abstract
```

Events are validated against the schema /schema/event, this schema include relations to /schema/who, /schema/localization, /schema/geo.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search