
Test RTD Documentation

Release 1.0.0

Alberto Pagliarini

October 23, 2015

1	Setup a frontend to consume API	3
1.1	Enable API on new frontend app	3
1.2	Enable API on old frontend app	3
2	Configuration	5
3	Response and Errors	7
3.1	Response	7
3.2	Errors	8
4	Authentication	9
4.1	Key concepts	9
4.2	Architecture	9
4.3	Customize authentication	10
5	Pagination	13
5.1	Define your API pagination default options	14
5.2	Paginate objects in custom endpoints	14
6	Customize endpoints	15
6.1	Custom endpoints	15
6.2	Blacklist endpoints	16
6.3	Enable special object types endpoints	16
6.4	Customize /objects endpoint with your own URL path filter types	16
7	Formatting BEdata objects	19
7.1	Introducing the ApiFormatter Component	19
7.2	Help ApiFormatter to cast object fields in the right way	19
7.3	Remove unwanted fields	21
7.4	Keep fields that are removed by default	22
8	API reference	23
8.1	Authentication /auth	23
26	subsection.8.1.4	
8.2	Objects /objects	26
8.3	Poster /poster	46
8.4	User profile /me	47
9	Markdown page	49

9.1 Code test	49
10 Indices and tables	51
HTTP Routing Table	53

BEdata frontend app can be easily enabled to serve REST API. Once enabled the API present a set of default endpoints that can be customized for frontend needs.

Setup a frontend to consume API

To use REST API in your frontend app you need at least BEdita 3.6.0 version. You can already also test it using 3-corylus branch.

Note: Because of authentication is handled using [Json Web Token \(IETF\)](#) and the JWT is digital signed using 'Security.salt' you should always remember to change it in app/config/core.php file:

```
Configure::write('Security.salt', 'my-security-random-string');
```

1.1 Enable API on new frontend app

- from shell

```
cd /path/to/bedita
./cake.sh frontend init
```

- in app/config/frontend.ini.php define \$config['api']['baseUrl'] with your API base url, for example

```
$config['api'] = array('baseUrl' => '/api/v1');
```

That's all! You are ready to consume the API!

Point the browser to your API base url and you should see the list of endpoints available, for example

```
{
  "auth": "http://example.com/api/v1/auth",
  "me": "http://example.com/api/v1/me",
  "objects": "http://example.com/api/v1/objects",
  "poster": "http://example.com/api/v1/poster"
}
```

1.2 Enable API on old frontend app

- create a new ApiController in your frontend

```
require(BEDITA_CORE_PATH . DS . 'controllers' . DS . 'api_base_controller.php');

class ApiController extends ApiBaseController {
```

```
//...  
}
```

- in `app/config/frontend.ini.php` define `$config['api']['baseUrl']` with your API base url.
- edit `app/config/routes.php` putting

```
$apiBaseUrl = Configure::read('api.baseUrl');  
if (!empty($apiBaseUrl) && is_string($apiBaseUrl)) {  
    Router::connect($apiBaseUrl . '/*', array('controller' => 'api', 'action' => 'route'));  
}
```

above

```
Router::connect('/', array('controller' => 'pages', 'action' => 'route'));
```

That's all!

After #570 we have implemented a new (and better) way to handle Exceptions. Remember to update your frontend `index.php` file:

```
if (isset($_GET['url']) && $_GET['url'] === 'favicon.ico') {  
    return;  
} else {  
    $Dispatcher = new Dispatcher();  
    $Dispatcher->dispatch();  
}
```

Also make sure you have defined `views/errors/error.tpl` in your frontend for generic error handling.

Configuration

To configure REST API you need to edit the frontend configuration file `app/config/frontend.ini.php`, for example

```
$config['api'] = array(
    'baseUrl' => '/api/v1',
    'allowedOrigins' => array(),
    'auth' => array(
        'component' => 'MyCustomAuth',
        'JWT' => array(
            'expiresIn' => 600,
            'alg' => 'HS256'
        ),
    ),
    'formatting' => array(
        'fields' => array(
            // fields that should be removed from results
            'remove' => array(
                'title',
                'Category' => array('name')
            ),
            // fields (removed by default) that should be kept
            'keep' => array(
                'ip_created',
                'Category' => array('object_type_id', 'priority')
            )
        ),
    ),
    'validation' => array(
        'writableObjects' => array('document', 'event')
    )
);
```

Possible configuration params are:

- `baseUrl` the base url of REST API. Every request done to `baseUrl` will be handled as an API REST request via routing rules
- `allowedOrigins` define which origins are allowed. Leave empty to allow all origins
- `auth` contains authentication configurations:
- `component` define the name of auth component to use. By default `ApiAuth` Component is used
- `JWT` define some options used in [Json Web Token](#) authentication as the “*expires in*” time (in seconds) and the hashing algorithm to use

- `formatting` permits to setup some formatting rules as object fields to *remove* or to *keep*
- `validation` setup some validation rules used generally in write operations. For example `writableObjects` define what object types are writable.

Response and Errors

3.1 Response

Usually the response of API query has the structure

```
{
  "api": "objects",
  "data": {},
  "method": "get",
  "params": [],
  "url": "https://example.com/api/v1/objects/1"
}
```

where:

- `api` is the endpoint called
- `data` is an object containing all data requested
- `method` is the HTTP verb used in the request
- `params` contains all query url params used in the request
- `url` is the complete url requested (full base url + basepath + endpoint + other)

To set data for response is available the method `ApiBaseController::setData()` that accepts an array as first argument. A second argument permits to replace (default) or merge present data with that passed.

Other meta data can be placed inside response object, for example `paging` useful to paginate results:

```
{
  "api": "objects",
  "data": {},
  "method": "get",
  "paging": {
    "page": 1,
    "page_size": 10,
    "page_count": 10,
    "total": 995,
    "total_pages": 100
  },
  "params": [],
  "url": "https://example.com/api/v1/objects/1/children"
}
```

where:

- `page` is the current page
- `page_size` is the page dimension
- `page_count` is the number of items inside current page
- `total` if the count of all items
- `total_pages` is the total pages available

Note: If you need to serve empty response body to client you can use `ApiBaseController::emptyResponse()` that by default send a **204 No Content** HTTP status code. Pass another status code as first argument to send different status code.

3.2 Errors

Every time the API thrown an error the response will be similar to

```
{
  "error": {
    "status": 405,
    "code": null,
    "message": "Method Not Allowed",
    "details": "Method Not Allowed",
    "more_info": null,
    "url": "https://example.com/api/v1/foobar"
  }
}
```

where:

- `status` is the HTTP status code
- `code` is the API error code (not implemented)
- `message` is the error message
- `details` is the error detail
- `more_info` is the url to error documentation (not implemented)
- `url` is the url that has responded with the error

Auhtentication

4.1 Key concepts

Access token A string granted by the authorization server used to identify the issuer of a request. The access token has to be sent to the resource server every time that the client want to access to protected resources.

BEdata REST API uses *JSON Web Token* as access token. It can be sent as `Authorization` HTTP header (preferred) using a `Bearer` scheme

```
Authorization: Bearer <token>
```

or as query string `/endpoint?access_token=<token>`

JSON Web Token, JWT JSON Web Tokens are an open, industry standard [RFC 7519](#) method for representing claims securely between two parties.

A JWT is composed by three parts:

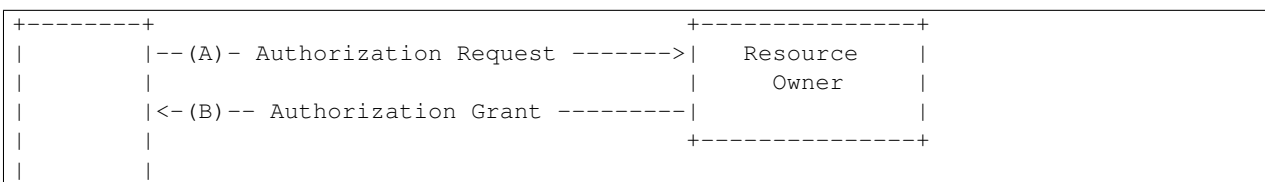
- an **header** containing informations about the token type and algorithm used. It is Base64URL encoded.
- a **payload** containing informations in the form of claims (informations we want to transmit). It is Base64URL encoded.
- a **signature** used to verify the authenticity of the JWT using an valid algorithm defined by [JSON Web Signature \(JWS\)](#) specification (for example a shared secret [HMAC](#)).

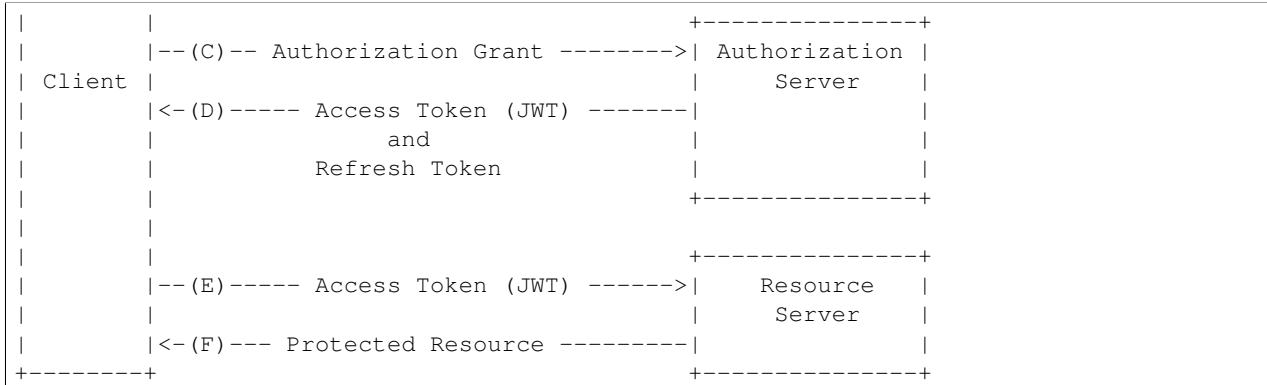
More info [here](#).

Refresh token An opaque token issued by the authorization server. It is useful to renew an expired *access token* without send the user credentials again. This token doesn't expire but can be revoked by `DELETE /auth/(string:refresh_token)`

4.2 Architecture

The API follow a **token based authentication flow** using a *JSON Web Token* as *access token* and an opaque token as *refresh token*.





Usually *JWT* payload contains the user `id` and some public claims, for example

```
{
  "iss": "https://example.com",
  "iat": "1441749523",
  "exp": "1441707000",
  "id": "15"
}
```

Important: Because of *JWT* is digital signed using `'Security.salt'` you should always remember to change it in `app/config/core.php` file:

```
Configure::write('Security.salt', 'my-security-random-string');
```

It is possible to invalidate all *access token* released simply changing that value.

By default all *GET* requests don't require client and user authentication unless the object requested has permission on it. In that case the user has to be authenticated before require the resource. Other operations as writing/deleting objects (*POST*, *PUT*, *DELETE* on `/objects` endpoint) are always protected instead and they always require authentication.

All the logic to handle authentication is in `ApiAuth` component and `ApiBaseController` use it for you so authentication works out of the box. If you need to protect *custom endpoints* you have to add to custom method

```
protected function customEndPoint() {
    if (!$this->ApiAuth->identify()) {
        throw new BeditaUnauthorizedException();
    }
}
```

4.3 Customize authentication

If you need to customize or change the authentication you can define your own auth component. To maintain the component method signature used in `ApiBaseController` your component should implements the interface `ApiAuthInterface`.

Remember that REST API are thought to implement token based authentication with the use of both `access_token` and `refresh_token` so the interface define methods to handle these tokens. If you need something different probably you would also override authentication methods of `ApiBaseController`.

In case you only need some little change it should be better to directly extend `ApiAuth` component that already implements the interface, and override the methods you need.

For example supposing you want to add additional check to user credentials, you can simply override `ApiAuth::authenticate()` method which deals with it:

```
App::import('Component', 'ApiAuth');

class CustomAuthComponent extends ApiAuthComponent {

    public function authenticate($username, $password, array $authGroupName = array()) {
        // authentication logic here
    }

}
```

and finally to activate the component all you have to do is define in configuration file `config/frontend.ini.php` the auth component you want to use.

```
$config['api'] = array(
    'baseUrl' => '/api',
    'auth' => array(
        'component' => 'CustomAuth'
    )
);
```

In `ApiController` you will have access to `CustomAuth` instance by `$this->ApiAuth` attribute.

Pagination

Requesting a list of objects by `/objects` endpoint the result will be paginated using default values that you can *customize* in `ApiController`.

In the response you'll see the pagination data in `paging` key

```
{
  "api": "objects",
  "data": {},
  "method": "get",
  "paging": {
    "page": 1,
    "page_size": 10,
    "page_count": 10,
    "total": 995,
    "total_pages": 100
  },
  "params": [],
  "url": "https://example.com/api/v1/objects/1/children"
}
```

where

- `page` is the current page
- `page_size` is the items per page
- `page_count` is the count of items in current page
- `total` is the total items
- `total_pages` is the total numbers of pages

To request a specific page simply call the endpoint passing `page` as GET parameter for example `/api/objects/1/children?page=5` to request the page 5.

You can also change the page size always through GET parameter, for example `/api/objects/1/children?page_size=50` to request 50 objects per page. `page_size` can't be greater of `$paginationOptions['maxPageSize']` defined in controller.

See below to know how to change the default values.

5.1 Define your API pagination default options

The default values used paginating items are defined in `ApiBaseController::paginationOptions` property.

```
protected $paginationOptions = array(  
    'page' => 1,  
    'pageSize' => 20,  
    'maxPageSize' => 100  
);
```

where `pageSize` is the default items per page and `maxPageSize` is the max page dimension that client can request. Requests with `page_size` greater of `maxPageSize` returns a 400 HTTP error.

If you want modify those defaults you can simply override that property in `ApiController`.

5.2 Paginate objects in custom endpoints

When a request has `page` or `page_size` as GET parameters those are validated and `$paginationOptions` is updated to contain the requested page options. A `dim` key equal to `pageSize` is added to be ready to use in some methods as `FrontendController::loadSectionObjects()`.

In this way in a 'custom' API endpoint you can simply do

```
protected function custom($id) {  
    $result = $this->loadSectionObjects($id, $this->paginationOptions);  
    // format and set pagination  
    $this->setPaging($this->ApiFormatter->formatPaging($result['toolbar']));  
  
    // do other stuff  
}
```

and you are sure that pagination will work properly without doing anything else.

Customize endpoints

6.1 Custom endpoints

Once you have [enabled a frontend to consume API](#) you have a set of [default available endpoints](#) visible pointing the browser to your API base url.

Sometimes you would want to define other endpoints to serve your custom data to clients. You can do it simply override the `$endpoints` attribute of `ApiBaseController`.

Write in your `ApiController`

```
protected $endpoints = array('friends');
```

and define the related custom method that will handle the data to show

```
protected function friends() {  
    $friendsList = array('Tom', 'Jerry');  
    $this->setData($friendsList);  
}
```

The `setData()` method takes care of put `$friendsList` array inside response data key. Point the browser to your API base url you should see 'friends' in the endpoints list and if you request `GET /api/base/url/friends` you should see

```
{  
    "api": "friends",  
    "data": [  
        "Tom",  
        "Jerry"  
    ],  
    "method": "get",  
    "params": [],  
    "url": "https://example.com/api/v1/friends"  
}
```

In this way all request types (GET, POST, PUT, DELETE) have to be handled by `friends()` method. Another possibility is to create one method for every request type allowed from the endpoint. It can be done creating methods named *"request type + endpoint camelized"*.

```
protected function getFriends() {  
}  
  
protected function postFriends() {  
}
```

```
protected function putFriends() {  
}  
  
protected function deleteFriends() {  
}
```

6.2 Blacklist endpoints

In some situations you will not want to expose some or all default endpoints, so in order to disable them it is possible to define a blacklist. After that calling those endpoints the response will be a **405 Method Not Allowed** HTTP error status code.

For example to blacklist “auth” and “objects” endpoints, in your ApiController override \$blacklistEndpoints writing

```
protected $blacklistEndpoints = array('auth', 'objects');
```

Now, pointing to API base url you shouldn’t see “auth” and “objects” endpoints anymore.

Pointing to them directly and you will receive a **405 HTTP error**.

6.3 Enable special object types endpoints

If you need you can also enable some special endpoint disabled by default. Those endpoints refer to BEdit object types mapping them to their pluralize form. So if you want to enable /documents end /galleries endpoints you have to edit ApiController

```
protected $whitelistObjectTypes = array('document', 'gallery');
```

These special endpoints automatically filter response objects through the object type related.

Again go to API base url to see ‘documents’ and ‘galleries’ added to endpoints list.

Note: Note that those special endpoints work only for GET requests.

6.4 Customize /objects endpoint with your own URL path filter types

objects endpoint can be customized with URL path filters building endpoint structured as /objects/:id/url_path_filter. URL path filters on by default are visible in ApiController::\$allowedObjectsUrlPath property

```
protected $allowedObjectsUrlPath = array(  
    'get' => array(  
        'relations',  
        'children',  
        'contents',  
        'sections',  
        'descendants',  
        'siblings',  
        //'ancestors',  
    ),  
);
```

```

        // 'parents'
    ),
    'post' => array(
        'relations',
        'children'
    ),
    'put' => array(
        'relations',
        'children'
    ),
    'delete' => array(
        'relations',
        'children'
    )
);

```

URL path filters can be inhibited or new ones can be added overriding that property in `ApiController`.

In practice URL path filters are divided by request type (GET, POST, ...) so it is possible doing request like GET `/objects/1/children`, POST `/objects/1/relations` but not POST `/objects/1/siblings` because of that filter is active only for GET requests.

Every URL path filter must have a corresponding controller method named “*request type + Objects + URL path filter camelized*” that will handle the request. First url part `:id` and every other url parts after URL path filter will be passed to that method as arguments.

For example, supposing to want to remove all ‘delete’ and ‘post’ URL path filters and add a new ‘foo_bar’ filter for GET request, in `ApiController` we can override

```

protected $allowedObjectsUrlPath = array(
    'get' => array(
        'relations',
        'children',
        'contents',
        'sections',
        'descendants',
        'siblings',
        'foo_bar'
    ),
);

```

and add the method

```

protected function getObjectsFooBar($objectId) {
    // handle request here
}

```

In this way the new URL path filter is active and reachable from GET `/objects/:id/foo_bar`. Every other request type (POST, PUT, DELETE) to that will receive **405 Method Not Allowed**.

If our ‘foo_bar’ URL path filter have to support GET `/objects/:id/foo_bar/:foo_val` requests then `ApiController::getObjectsFooBar()` will receive `:foo_val` as second argument. A best practice should be to add to method a validation on the number of arguments supported to avoid to respond to request as GET `/objects/:id/foo_bar/:foo_val/bla/bla/bla`.

```

protected function getObjectsFooBar($objectId, $fooVal = null) {
    if (func_num_args() > 2) {
        throw new BeditaBadRequestException();
    }
}

```

```
} // handle request here
```

Formatting BEdita objects

7.1 Introducing the ApiFormatter Component

To respond with consistent data the BEdita object types are transformed and formatted using the `ApiFormatter` Component that deals with cleaning objects from useless data and casting and transforming some fields in correct format.

If you have a look at `/objects/:id` response you'll see that fields as `'id'` are **integer** other like `'latitude'` and `'longitude'` of geo tag are **float** and **dates are formatted in ISO-8601**. `ApiFormatter` Component with a little help from Models takes care of it.

When you load an object or list of objects you should always use the `ApiFormatter` Component to have data always formatted in the same way.

```
// load an object
$object = $this->loadObj($id);
$result = $this->ApiFormatter->formatObject($object);
// in $result['object'] you have the formatted object

// list of objects
$objects = $this->loadSectionObjects($id, array('itemsTogether' => true));
$result = $this->ApiFormatter->formatObjects($objects['children']);
// in $result['objects'] you have the formatted objects
```

`ApiFormatter::formatObject()` and `ApiFormatter::formatObjects()` accept as second argument an array of options with which it is possible add to the formatted object the count of relations and children.

```
$result = $this->ApiFormatter->formatObject($object, array(
    'countRelations' => true,
    'countChildren' => true
));
```

By default no count is done.

7.2 Help ApiFormatter to cast object fields in the right way

When formatting BEdita object `ApiFormatter` asks help to related object type Model to know which fields have to be cast in the right type. Basically every object type returns an array of fields that are defined in database as `'integer'`, `'float'`, `'date'`, `'datetime'`, `'boolean'`. This array is returned from `BEAppObjectModel::apiTransformer()` method and it is something similar to

```
array(  
  'id' => 'integer',  
  'start_date' => 'datetime',  
  'end_date' => 'datetime',  
  'duration' => 'integer',  
  'object_type_id' => 'integer',  
  'created' => 'datetime',  
  'modified' => 'datetime',  
  'valid' => 'boolean',  
  'user_created' => 'integer',  
  'user_modified' => 'integer',  
  'fixed' => 'boolean',  
  'GeoTag' => array(  
    'id' => 'integer',  
    'object_id' => 'integer',  
    'latitude' => 'float',  
    'longitude' => 'float',  
    'gmaps_lookat' => array(  
      'latitude' => 'float',  
      'longitude' => 'float',  
      'zoom' => 'integer',  
    )  
  )  
  ,  
  'Tag' => array(  
    'id' => 'integer',  
    'area_id' => 'integer',  
    'object_type_id' => 'integer',  
    'priority' => 'integer',  
    'parent_id' => 'integer',  
  )  
  ,  
  'Category' => array(  
    'id' => 'integer',  
    'area_id' => 'integer',  
    'object_type_id' => 'integer',  
    'priority' => 'integer',  
    'parent_id' => 'integer',  
  )  
)
```

By default only tables that form the object chain plus ‘categories’, ‘tags’ and ‘geo_tags’ are automatically returned, but that method can be overridden to customize the result. For example the Event model add to basic transformer the DateItem transformer:

```
public function apiTransformer(array $options = array()) {  
    $transformer = parent::apiTransformer($options);  
    $transformer['DateItem'] = $this->DateItem->apiTransformer($options);  
    return $transformer;  
}
```

The ApiFormatter uses these transformers merged to common object transformer ApiFormatterComponent::\$transformers['object'] to present consistent data to client. It is possible to use some special transformer types that are:

- underscoreField that underscorize a camelcase field maintaining value unchanged
- integerArray that cast to integer all array values

7.3 Remove unwanted fields

Another useful task of `ApiFormatter` is to clean unwanted fields from data exposed to client. To do that it uses `ApiFormatter::$objectFieldsToRemove` array that can be customized through configuration or on the fly in controller.

7.3.1 Add fields to remove from configuration

In `config/frontend.ini.php` or `config/frontend.cfg.php` is possible to customize which fields exposed by default you want to remove from results.

```
$config['api'] = array(
    'baseUrl' => '/api/v1',
    ...
    'formatting' => array(
        'fields' => array(
            // fields that should be added
            // to ApiFormattingComponent::objectFieldsToRemove
            // i.e. removed from formatted object
            'remove' => array(
                'description',
                'title',
                'Category' => array('name'),
                'GeoTag' => array('title'),
                'Tag'
            )
        )
    )
);
```

In this way you say to `ApiFormatter` that ‘description’, ‘title’, ‘name’ of ‘Category’, ‘title’ of ‘GeoTag’ and all ‘Tag’ array must be cleaned from final results. Every time `ApiFormatter::formatObject()` or `ApiFormatter::formatObjects()` is called the data are cleaned up using `ApiFormatter::cleanObject()`.

7.3.2 Add fields to remove on the fly

In your `ApiController` you can decide in every moment to change which fields remove from results using `ApiFormatter::objectFieldsToRemove()` method.

```
// get the current value
$currentFieldsToRemove = $this->ApiFormatter->objectFieldsToRemove();

// to override all. It completely replaces current fields to remove with new one
$this->ApiFormatter->objectFieldsToRemove(
    array(
        'title',
        'description'
    ),
    true
);

// to add new fields to remove
$this->ApiFormatter->objectFieldsToRemove(array(
```

```
'remove' => array('title', 'description')
));
```

7.4 Keep fields that are removed by default

Sometime you could want to present to client some fields that normally are cleaned up. Likewise to what seen with fields to remove, it is possible do it from configuration or on the fly.

7.4.1 Add fields to keep from configuration

In config/frontend.cfg.php

```
$config['api'] = array(
    'baseUrl' => '/api/v1',
    ...
    'formatting' => array(
        'fields' => array(
            // fields that should be removed
            // to ApiFormattingComponent::objectFieldsToRemove
            // i.e. kept in formatted object
            'keep' => array(
                'fixed',
                'ip_created',
                'Category' => array('object_type_id', 'priority')
            )
        )
    )
);
```

In this way you say to ApiFormatter that ‘fixed’, ‘ip_created’ and ‘object_type_id’, ‘priority’ of ‘Category’ must be preserved and presented to client.

7.4.2 Add fields to keep on the fly

In your ApiController

```
// to keep fields
$this->ApiFormatter->objectFieldsToRemove(array(
    'keep' => array('ip_created', 'fixed')
));
```

It is possible to mix ‘remove’ and ‘keep’ options both in configuration and in controller.

API reference

A frontend app enabled to consume REST API exposes a set of default endpoints.

Note: Every `POST` request can send the payload as `x-www-form-urlencoded` or `application/json`. For readability all examples will use `Content-type: application/json`.

8.1 Authentication /auth

It used to retrieve an *access token* to access protected items, renew *access token* and remove permissions. The *access token* is a *Json Web Token* (IETF). More info on [authentication](#)

Important: Because of *JWT* is digital signed using `'Security.salt'` you should always remember to change it in `app/config/core.php` file:

```
Configure::write('Security.salt', 'my-security-random-string');
```

It is possible to invalidate all *access token* released simply changing that value.

8.1.1 Obtain an access token

`POST /auth`

Authenticate an user to obtain an *access token*.

Request JSON Object

- **username** (*string*) – the username
- **password** (*string*) – the password
- **grant_type** (*string*) – “password”, the grant type to apply (password is the default, it can be omitted)

Response Headers

- **Content-Type** – `application/json`

Status Codes

- **200 OK** – response contains *access token* and *refresh token*
- **400 Bad Request** – when required parameters are missing or the request is malformed

- 401 Unauthorized – when authentication fails

Example request:

```
POST /auth HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json

{
  "username": "test",
  "password": "test",
  "grant_type": "password"
}
```

Example response:

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

{
  "api": "auth",
  "data": {
    "access_token": "eyJ0eXAiOi...",
    "expires_in": 600,
    "refresh_token": "51a3f718e7abc712e421f64ea497a323aea4e76f"
  },
  "method": "post",
  "params": [ ],
  "url": "https://example.com/api/auth"
}
```

Note: Once you received the access token you have to use it in every request that requires authentication. It can be used in HTTP header

```
Authorization: Bearer <token>
```

or as query string /api/endpoint?access_token=<token>

8.1.2 Renew the access token

If the access token was expired you need to generate a new one started by refresh token. **In this case do not pass the expired access token**

POST /auth

Renew an *access_token*.

Request JSON Object

- **refresh_token** (*string*) – the *refresh token* to use to renew *access token*
- **grant_type** (*string*) – “*refresh_token*”, the grant type to apply

Response Headers

- **Content-Type** – application/json

Status Codes

- **200 OK** – Success, it responds with the new *access token* and *refresh token*

- 400 Bad Request – when required parameters are missing or the request is malformed
- 401 Unauthorized – when *refresh token* is invalid

Example request:

```
POST /auth HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json

{
  "grant_type": "refresh_token",
  "refresh_token": "51a3f718e7abc712e421f64ea497a323aea4e76f"
}
```

Example response:

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

{
  "api": "auth",
  "data": {
    "access_token": "rftJasd3.....",
    "expires_in": 600,
    "refresh_token": "51a3f718e7abc712e421f64ea497a323aea4e76f"
  },
  "method": "post",
  "params": [ ],
  "url": "https://example.com/api/auth"
}
```

8.1.3 Get the updated time to access token expiration

GET /auth

It returns the updated `expires_in` time for *access token*

Request Headers

- Authorization – the *access token* as Bearer token

Response Headers

- Content-Type – application/json

Status Codes

- 200 OK – no error, payload contains the updated `expires_in` value
- 400 Bad Request – the request is malformed
- 401 Unauthorized – the *access token* is invalid

Example request:

```
GET /auth HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

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

{
  "api": "auth",
  "data": {
    "access_token": "rftJasd3....",
    "expires_in": 48
  },
  "method": "get",
  "params": [ ],
  "url": "https://example.com/api/auth"
}
```

8.1.4 Revoking a refresh token

In order to invalidate an *access token* you need to remove it from client and revoke the *refresh token*

DELETE /auth/ (string: *refresh_token*)

Revoke a *refresh token*

Request Headers

- *Authorization* – the *access token* as Bearer token

Parameters

- **refresh_token** (string) – the *refresh token* to revoke

Status Codes

- 204 No Content – the refresh token was deleted
- 400 Bad Request – the request is malformed
- 401 Unauthorized – the *access token* is invalid or
- 404 Not Found – the *refresh token* was already revoked or not exists

8.2 Objects /objects

8.2.1 Get an object

GET /objects/ (object_id)

Get an object detail.

Request Headers

- *Authorization* – optional *access token*

Parameters

- **object_id** (int|string) – identify a BEdita object. It can be the object id or the object unique name (nickname)

Response Headers

- *Content-Type* – application/json

Status Codes

- 200 OK – Success
- 400 Bad Request – Malformed request
- 401 Unauthorized – The object *object_id* is protected and the request is not authorized
- 403 Forbidden – The request is authorized but without sufficient permission to access object *object_id*
- 404 Not Found – Object *object_id* not found

Note: Note that the response data fields can change depending of BEdita object type exposed and configuration.

Example request:

```
GET /objects/15 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

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

{
  "api": "objects",
  "data": {
    "object": {
      "id": 15,
      "start_date": "2015-01-08T00:00:00+0100",
      "end_date": null,
      "subject": null,
      "abstract": null,
      "body": "This is the body text",
      "object_type_id": 22,
      "created": "2015-01-30T10:04:49+0100",
      "modified": "2015-05-08T12:59:49+0200",
      "title": "hello world",
      "nickname": "hello-world",
      "description": "the description",
      "valid": true,
      "lang": "eng",
      "rights": "",
      "license": "",
      "creator": "",
      "publisher": "",
      "note": null,
      "comments": "off",
      "publication_date": "2015-01-08T00:00:00+0100",
      "languages": {
        "ita": {
          "title": "ciao mondo"
        }
      },
      "relations": {
        "attach": {
          "count": 8,
          "url": "https://example.com/api/objects/15/relation/attach"
        }
      }
    }
  }
}
```

```
    },
    "seealso": {
      "count": 2,
      "url": "https://example.com/api/objects/15/relation/seealso"
    }
  },
  "object_type": "Document",
  "authorized": true,
  "free_access": true,
  "custom_properties": {
    "bookpagenumber": "12",
    "number": "8"
  },
  "geo_tags": [
    {
      "id": 68799,
      "object_id": 218932,
      "latitude": 44.4948179,
      "longitude": 11.33969,
      "address": "via Rismondo 2, Bologna",
      "gmaps_lookats": {
        "zoom": 16,
        "mapType": "k",
        "latitude": 44.4948179,
        "longitude": 11.33969,
        "range": 44052.931589613
      }
    }
  ],
  "tags": [
    {
      "label": "tag one",
      "name": "tag-one"
    },
    {
      "label": "tag two",
      "name": "tag-two"
    }
  ],
  "categories": [
    {
      "id": 266,
      "area_id": null,
      "label": "category one",
      "name": "category-one"
    },
    {
      "id": 323,
      "area_id": null,
      "label": "category two",
      "name": "category-two"
    }
  ]
},
"method": "get",
"params": [],
"url": "https://example.com/api/objects/15"
```



```
}
```

Note: Every object can have relations with other objects. The count of objects related is in `data.object.relations.<relation_name>` where there are `count` (the number of related objects) and `url` fields. The `url` is the complete API request url to get the object related, for example <https://example.com/api/objects/15/relations/attach>

If `object_id` corresponds to a **section** or a **publication** then the response will contain `data.object.children` with the total count of children, count of contents, count of sections and the related url.

```
{
  "children": {
    "count": 14,
    "url": "https://example.com/api/objects/1/children",
    "contents": {
      "count": 12,
      "url": "https://example.com/api/objects/1/contents"
    },
    "sections": {
      "count": 2,
      "url": "https://example.com/api/objects/1/sections"
    }
  }
}
```

8.2.2 Get a collection of objects

The `/objects` endpoint can be used to retrieve a collection of objects.

Get publication's descendants

GET /objects

Return a paginated list of objects that are descendants of the related publication configured in `app/config/frontend.ini.php`. The response will be an array of objects as shown below.

Request Headers

- `Authorization` – optional *access token*

Response Headers

- `Content-Type` – `application/json`

Status Codes

- `200 OK` – Success
- `400 Bad Request` – Malformed request
- `401 Unauthorized` – The request is not authorized to access to protected publication
- `403 Forbidden` – The request is authorized but without sufficient permissions to access to protected publication

Example request:

```
GET /objects HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

For readability the fields of objects are limited to “title” but they are similar to `GET /objects/(object_id)` example

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

{
  "api": "objects",
  "data": {
    "objects": [
      {
        "id": 2,
        "title": "title one"
      },
      {
        "id": 3,
        "title": "title two"
      },
      {
        "id": 4,
        "title": "title three"
      },
      {
        "id": 5,
        "title": "title four"
      },
      {
        "id": 6,
        "title": "title five"
      }
    ]
  },
  "method": "get",
  "paging": {
    "page": 1,
    "page_size": 5,
    "page_count": 5,
    "total": 50,
    "total_pages": 10
  },
  "params": [],
  "url": "https://example.com/api/objects/1/children"
}
```

Get object’s children

GET `/objects/(object_id)/children`

Return the paginated children of object *object_id*. The object has to be a section or the publication.

Request Headers

- **Authorization** – optional *access token*

Parameters

- **object_id** (*int|string*) – identify a BEdita object. It can be the object id or the object unique name (nickname)

Response Headers

- **Content-Type** – application/json

Status Codes

- **200 OK** – Success
- **400 Bad Request** – Malformed request
- **401 Unauthorized** – The object *object_id* is protected and the request is not authorized
- **403 Forbidden** – The request is authorized but without sufficient permission to access object *object_id*
- **404 Not Found** – Object *object_id* not found

Get object's children of type *section***GET** `/objects/ (object_id) /sections`

Return the paginated children of object *object_id*. The object has to be a section or the publication. The children are just sections (*section BEdita object type*)

Request Headers

- **Authorization** – optional *access token*

Parameters

- **object_id** (*int|string*) – identify a BEdita object. It can be the object id or the object unique name (nickname)

Response Headers

- **Content-Type** – application/json

Status Codes

- **200 OK** – Success
- **400 Bad Request** – Malformed request
- **401 Unauthorized** – The object *object_id* is protected and the request is not authorized
- **403 Forbidden** – The request is authorized but without sufficient permission to access object *object_id*
- **404 Not Found** – Object *object_id* not found

Get object's children of type *contents***GET** `/objects/ (object_id) /contents`

Return the paginated children of object *object_id*. The object has to be a section or the publication. The children are other than sections.

Request Headers

- **Authorization** – optional *access token*

Parameters

- **object_id** (*int|string*) – identify a BEdita object. It can be the object id or the object unique name (nickname)

Response Headers

- **Content-Type** – application/json

Status Codes

- **200 OK** – Success
- **400 Bad Request** – Malformed request
- **401 Unauthorized** – The object *object_id* is protected and the request is not authorized
- **403 Forbidden** – The request is authorized but without sufficient permission to access object *object_id*
- **404 Not Found** – Object *object_id* not found

Get object's descendants

GET /objects/ (*object_id*) /descendants

Return the paginated children of object *object_id*. The object has to be a section or the publication. The children are other than sections.

Request Headers

- **Authorization** – optional *access token*

Parameters

- **object_id** (*int|string*) – identify a BEdita object. It can be the object id or the object unique name (nickname)

Response Headers

- **Content-Type** – application/json

Status Codes

- **200 OK** – Success
- **400 Bad Request** – Malformed request
- **401 Unauthorized** – The object *object_id* is protected and the request is not authorized
- **403 Forbidden** – The request is authorized but without sufficient permission to access object *object_id*
- **404 Not Found** – Object *object_id* not found

Get object's siblings

GET /objects/ (*object_id*) /siblings

Return the paginated siblings of object *object_id*.

Request Headers

- **Authorization** – optional *access token*

Parameters

- **object_id** (*int|string*) – identify a BEdita object. It can be the object id or the object unique name (nickname)

Response Headers

- **Content-Type** – application/json

Status Codes

- **200 OK** – Success
- **400 Bad Request** – Malformed request
- **401 Unauthorized** – The object *object_id* is protected and the request is not authorized
- **403 Forbidden** – The request is authorized but without sufficient permission to access object *object_id*
- **404 Not Found** – Object *object_id* not found

Get relations count**GET /objects/ (*object_id*) /relations**

Returns a summary of relations information about object *object_id*. It shows every relation with the **count** and the **url** to get the related objects detail.

Request Headers

- **Authorization** – optional *access token*

Parameters

- **object_id** (*int|string*) – identify a BEdita object. It can be the object id or the object unique name (nickname)

Response Headers

- **Content-Type** – application/json

Status Codes

- **200 OK** – Success
- **400 Bad Request** – Malformed request
- **401 Unauthorized** – The object *object_id* is protected and the request is not authorized
- **403 Forbidden** – The request is authorized but without sufficient permission to access object *object_id*
- **404 Not Found** – Object *object_id* not found

Example request:

```
GET /objects/15/relations HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

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

{
  "api": "objects",
  "data": {
    "attach": {
      "count": 1,
      "url": "https://example.com/api/objects/1/relations/attach"
    },
    "seealso": {
      "count": 2,
      "url": "https://example.com/api/objects/1/relations/seealso"
    }
  },
  "method": "get",
  "params": [],
  "url": "https://example.com/api/objects/1/relations"
}
```

Get the related objects detail

GET `/objects/ (object_id) /relations/`

string: *relation_name* Return the paginated collection of objects related by *relation_name* to object *object_id*.

Request Headers

- **Authorization** – optional *access token*

Parameters

- **object_id** (*int|string*) – identify a BEdita object. It can be the object id or the object unique name (nickname)

Response Headers

- **Content-Type** – application/json

Status Codes

- 200 OK – Success
- 400 Bad Request – Malformed request
- 401 Unauthorized – The object *object_id* is protected and the request is not authorized
- 403 Forbidden – The request is authorized but without sufficient permission to access object *object_id*
- 404 Not Found – Object *object_id* not found

Get the relation detail between objects

GET `/objects/ (object_id) /relations/`

string: *relation_name* / **int:** *related_id* Returns the relation detail between object *object_id* and *related_id*.

Request Headers

- **Authorization** – optional *access token*

Parameters

- **object_id** (*int|string*) – identify a BEdita object. It can be the object id or the object unique name (nickname)
- **relation_name** (*string*) – the name of the relation that ties *object_id* and *related_id*
- **related_id** (*int*) – the object id of the related object

Response Headers

- **Content-Type** – application/json

Status Codes

- **200 OK** – Success
- **400 Bad Request** – Malformed request
- **401 Unauthorized** – The object *object_id* is protected and the request is not authorized
- **403 Forbidden** – The request is authorized but without sufficient permission to access object *object_id*
- **404 Not Found** – Object *object_id* not found

Example request:

```
GET /objects/15/relations/attach/23 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

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

{
  "api": "objects",
  "data": {
    "priority": 3,
    "params": {
      "label": "here the label"
    }
  },
  "method": "get",
  "params": [],
  "url": "https://example.com/api/objects/1/relations/attach/2"
}
```

Get the child position

GET /objects/ (*object_id*) /children/

int: *child_id* Return the position (*priority* key) of *child_id* relative to all children of parent object *object_id*

Request Headers

- **Authorization** – optional *access token*

Parameters

- **object_id** (*int|string*) – identify a BEdita object. It can be the object id or the object unique name (nickname)
- **child_id** (*int*) – the object id of the child of object *object_id*

Response Headers

- **Content-Type** – application/json

Status Codes

- **200 OK** – Success
- **400 Bad Request** – Malformed request
- **401 Unauthorized** – The object *object_id* is protected and the request is not authorized
- **403 Forbidden** – The request is authorized but without sufficient permission to access object *object_id*
- **404 Not Found** – Object *object_id* not found

Example request:

```
GET /objects/1/children/2 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

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

{
  "api": "objects",
  "data": {
    "priority": 3
  },
  "method": "get",
  "params": [],
  "url": "https://example.com/api/objects/1/children/2"
}
```

8.2.3 Create an object

POST /objects

Create a new BEdita object type.

Important: To write an object it has to be [configured to be writable](#)

```
$config['api'] = array(
    // ....
    'validation' => array(
        // to save 'document' and 'event' object types
        'writableObjects' => array('document', 'event')
    )
);
```

The request body has to be a JSON as


```
{
  "data": {}
}
```

where inside "data" are placed all fields to save. User has to be [authenticated](#) and "data": {} must contain: `object_type` i.e. the object type you want to create at least a parent (`parents` key) accessible (with right permission for user authorized) on publication tree or at least a relation (`relations` key) with another object reachable (where *reachable* means an accessible object on tree or related to an accessible object on tree).

Required keys in JSON are shown below.

Request Headers

- [Authorization](#) – (required) *access token*

Request JSON Object

- **data.object_type** (*string*) – (required) the object type to create
- **data.parents** (*array*) – (required if `data.relations` with conditions specified below missing) a list of parents. Parents must be accessible (with right permission for user authorized) on publication tree
- **data.relations** (*object*) – (required if `data.parents` with conditions specified above missing) an object of relations where the keys are the relations' names. Every relation contains an array of objects of `related_id` and optionally of relation params

```
{
  "name1": [
    {
      "related_id": 1
    },
    {
      "related_id": 2,
      "params": {
        "name_param_one": "value param one",
        "name_param_two": "value param two"
      }
    }
  ],
  "name2": [
    {
      "related_id": 3
    }
  ]
}
```

Response Headers

- [Content-Type](#) – `application/json`
- [Location](#) – The url to the resource just created
`https://example.com/objects/(object_id)`

Status Codes

- [201 Created](#) – Success, the object was created. Return the object detail as in [GET /objects/\(object_id\)](#)
- [400 Bad Request](#) – Required parameters are missing or the request is malformed

- 401 Unauthorized – Request is not authorized

Example request:

```
POST /objects HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json

{
  "data": {
    "title": "My title",
    "object_type": "event",
    "description": "bla bla bla",
    "parents": [1, 34, 65],
    "relations": {
      "attach": [
        {
          "related_id": 12,
          "params": {
            "label": "foobar"
          }
        },
        {
          "related_id": 23
        }
      ],
      "seealso": [
        {
          "related_id": 167
        }
      ]
    },
    "categories": ["name-category-one", "name-category-two"],
    "tags": ["name-tag-one", "name-tag-two"],
    "geo_tags": [
      {
        "title": "geo tag title",
        "address": "via ...",
        "latitude": 43.012,
        "longitude": 10.45
      }
    ],
    "date_items": [
      {
        "start_date": "2015-07-08T15:00:35+0200",
        "end_date": "2015-07-08T15:00:35+0200",
        "days": [0, 3, 4]
      },
      {
        "start_date": "2015-09-01T15:00:35+0200",
        "end_date": "2015-09-30T15:00:35+0200"
      }
    ]
  }
}
```

Example response:

```

HTTP/1.1 201 Created
Content-Type: application/json

{
  "api": "objects",
  "data": {
    "id": 45,
    "title": "My title",
    "object_type": "event",
    "description": "bla bla bla"
  },
  "method": "post",
  "params": [],
  "url": "https://example.com/api/objects"
}

```

The response payload contains the created object detail. *In the example above only some fields are shown.* dates must be in ISO 8601 format.

8.2.4 Update an object

POST /objects

Update an existent object.

Important: To write an object it has to be [configured to be writable](#)

```

$config['api'] = array(
    // ....
    'validation' => array(
        // to save 'document' and 'event' object types
        'writableObjects' => array('document', 'event')
    )
);

```

POST request can be also used to **update an existent object**. In that case the object id has to be passed in "data" (as *creating object*) and object_type can be omitted.

Request Headers

- **Authorization** – (required) *access token*

Request JSON Object

- **data.id** (*string*) – (required) the id of the object to update

Response Headers

- **Content-Type** – application/json

Status Codes

- **200 OK** – Success, the object was updated. Return the object detail as in *GET /objects/(object_id)*
- **400 Bad Request** – Required parameters are missing or the request is malformed
- **401 Unauthorized** – Request is not authorized

8.2.5 Create or update relations between objects

POST `/objects/(object_id)/relations/`

string: *relation_name* Create relations *relation_name* between *object_id* and other objects. If the relation between objects already exists then it will be updated.

Request data must be an array of relation data or only a relation data if you need to save only one relation.

Request Headers

- **Authorization** – (required) *access token*

Request JSON Array of Objects

- **related_id** (*string*) – (required) the related object id
- **params** (*string*) – (optional) it depends from relation type
- **priority** (*string*) – (optional) is the position of the relation. Relation with lower priority are shown before.

Response Headers

- **Content-Type** – application/json
- **Location** – If at least a new relation was created (201 Created). The url to *collection of related objects*

Status Codes

- 200 OK – Success but no new relation was created.
- 201 Created – Success and at least a new relation was created. Return the object detail as in `GET /objects/(object_id)`
- 400 Bad Request – Required parameters are missing or the request is malformed
- 401 Unauthorized – Request is not authorized

Example request to create/update only one relation:

```
POST /objects/3/relations/attach HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json

{
  "data": {
    "related_id": 34,
    "priority": 3
  }
}
```

Example request to create/update a bunch of relations:

```
POST /objects/3/relations/attach HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json

{
  "data": [
    {
      "related_id": 15,
```

```

        "params": {
            "label": "my label"
        },
        {
            "related_id": 28
        }
    ]
}

```

Example response:

```

HTTP/1.1 201 Created
Host: example.com
Location: https://example.com/objects/3/relations/attach
Accept: application/json, text/javascript
Content-Type: application/json

```

The response body will be the same as `GET /objects/(object_id)/relations/(string:relation_name)`

8.2.6 Replace relation data between objects

PUT /objects/(object_id)/relations/

string: *relation_name*/**int:** *related_id* Replace the relation *relation_name* data between *object_id* and *related_id*.

Request Headers

- **Authorization** – (required) *access token*

Request JSON Object

- **related_id** (*string*) – (required) the related object id
- **params** (*string*) – (optional) it depends from relation type
- **priority** (*string*) – (optional) is the position of the relation. Relation with lower priority are shown before.

Response Headers

- **Content-Type** – application/json

Status Codes

- 200 OK – Success
- 400 Bad Request – Required parameters are missing or the request is malformed
- 401 Unauthorized – Request is not authorized

At least params or priority must be defined. If one of these is not passed it will be set to null.

Example request:

```

PUT /objects/1/relations/attach/2 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json

{
    "data": {

```

```
"priority": 3,
"params": {
  "label": "new label"
}
}
```

Example response:

```
HTTP/1.1 200 Success
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json
```

The response body will be the same as `GET /objects/(object_id)/relations/(string:relation_name)/(in`

8.2.7 Add or edit children

POST /objects/(object_id)/children

Add or edit children to area/section object type identified by *object_id*

Request data must be an array of child data or only a child data if you need to save only one child.

Request Headers

- **Authorization** – (required) *access token*

Request JSON Array of Objects

- **child_id** (*string*) – (required) the child object id
- **priority** (*string*) – (optional) is the position of the child on the tree. Relation with lower priority are shown before.

Response Headers

- **Content-Type** – application/json
- **Location** – If at least a new child was created (201 Created) it contains the url to *collection of children objects*.

Status Codes

- **200 OK** – Success but all objects already were children of *object_id*. The children position (priority) could be changed. Response body is the children detail `GET /objects/(object_id)/children`
- **201 Created** – Success and at least a new child was added to parent *object_id*. Response body is the children detail `GET /objects/(object_id)/children`.
- **400 Bad Request** – Required parameters are missing or the request is malformed
- **401 Unauthorized** – Request is not authorized

Example request to add/edit many children:

```
POST /objects/3/children HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json

{
```

```

    "data": [
      {
        "child_id": 15,
        "priority": 3
      },
      {
        "child_id": 28
      }
    ]
  }

```

Example request to add/edit one child:

```

POST /objects/3/children HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json

{
  "data": {
    "child_id": 34,
    "priority": 3
  }
}

```

The response body will be the same as `GET /objects/(object_id)/children`

8.2.8 Update child position

PUT `/objects/(object_id)/children/`

int: `child_id` Change the child position inside the children of `object_id`.

Request Headers

- **Authorization** – (required) *access token*

Request JSON Object

- **priority** (*string*) – (required) the position of child object id

Response Headers

- **Content-Type** – `application/json`

Status Codes

- **200 OK** – Success. Children position (`priority`) updated.
- **400 Bad Request** – Required parameters are missing or the request is malformed
- **401 Unauthorized** – Request is not authorized

Example request:

```

POST /objects/1/children/2 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json

{
  "data": {

```

```
    "priority": 5
  }
}
```

Example response:

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

{
  "api": "objects",
  "data": {
    "priority": 5
  },
  "method": "get",
  "params": [],
  "url": "https://example.com/api/objects/1/children/2"
}
```

8.2.9 Delete an object

DELETE `/objects/` (*object_id*)Delete the object *object_id***Request Headers**

- **Authorization** – (required) *access token*

Response Headers

- **Content-Type** – application/json

Status Codes

- **204 No Content** – The object was deleted.
- **400 Bad Request** – Required parameters are missing or the request is malformed
- **401 Unauthorized** – Request is not authorized
- **404 Not Found** – The object to delete not exists or it has already been removed

Example request:

```
DELETE /objects/15 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

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

8.2.10 Delete a relation between objects

DELETE `/objects/` (*object_id*) `/relations/`**string:** *relation_name* **int:** *related_id* Delete the relation *relation_name* between *object_id* and *related_id***Request Headers**

- **Authorization** – (required) *access token*

Response Headers

- **Content-Type** – application/json

Status Codes

- **204 No Content** – The relation *relation_name* between *object_id* and *related_id* was deleted.
- **400 Bad Request** – Required parameters are missing or the request is malformed
- **401 Unauthorized** – Request is not authorized
- **404 Not Found** – The relation *relation_name* between *object_id* and *related_id* not exists or it has already been removed

Example request:

```
DELETE /objects/10/relations/seealso/36 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

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

8.2.11 Remove a child from a parent

DELETE /objects/ (*object_id*) /children/
int: *child_id* Remove *child_id* from *object_id* children

Request Headers

- **Authorization** – (required) *access token*

Response Headers

- **Content-Type** – application/json

Status Codes

- **204 No Content** – *child_id* was removed from *object_id* children.
- **400 Bad Request** – Required parameters are missing or the request is malformed
- **401 Unauthorized** – Request is not authorized
- **404 Not Found** – *child_id* not exists or it has already been removed from *object_id* children.

Example request:

```
DELETE /objects/1/children/3 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

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

8.3 Poster /poster

Warning: This endpoint is highly volatile. It could be completely changed or removed in next versions.

8.3.1 Get the image representation of object *object_id* as thumbnail url

GET /poster/ (*object_id*)

Get the thumbnail url of an image representation of the object *object_id*. The thumbnail returned depends from the object type of *object_id* and from its relations, in particular:

- 1.if object *object_id* has a 'poster' relation with an image object then it returns a thumbnail of that image
- 2.else if the object is an image then it returns a thumbnail of the object
- 3.else if the object has an 'attach' relation with an image object then it returns a thumbnail of that image

Request Headers

- **Authorization** – optional *access token*

Parameters

- **object_id** (*int|string*) – identify a BEdita object. It can be the object id or the object unique name (nickname)

Query Parameters

- **width** (*int*) – the thumbnail width
- **height** (*int*) – the thumbnail height

Response Headers

- **Content-Type** – application/json

Status Codes

- **200 OK** – Success
- **401 Unauthorized** – The object *object_id* is protected and the request is not authorized
- **403 Forbidden** – The request is authorized but without sufficient permission to access object *object_id*
- **404 Not Found** – Object *object_id* not found

Example request:

```
GET /poster/5 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json
```

Example response:

```
HTTP/1.1 200 Success
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json
```

```
{
  "api": "poster",
  "data": {
    "id": 5,
    "uri": "https://media.server/path/to/thumb/thumbnail.jpg"
  },
  "method": "get",
  "params": [],
  "url": "https://example.com/api/poster/5"
}
```

8.4 User profile /me

Warning: This endpoint is highly volatile. It could be completely changed or removed in next versions.

8.4.1 Obtain information about authenticated user

GET /me

Return information about current authenticated user

Request Headers

- **Authorization** – (required) *access token*

Response Headers

- **Content-Type** – application/json

Status Codes

- **200 OK** – Success
- **401 Unauthorized** – The request is not authorized
- **404 Not Found** –

See also:

To an index of all API requests see HTTP Routing Table

Markdown page

Ciao pagina del caz in **markdown!**

- uno
- due
- tre

9.1 Code test

In line code

```
$x = 1;  
echo $x;
```

Indices and tables

- `genindex`
- HTTP Routing Table

/auth

GET /auth, 25
POST /auth, 24
DELETE /auth/(string:refresh_token), 26

/me

GET /me, 47

/objects

GET /objects, 29
GET /objects/(object_id), 26
GET /objects/(object_id)/children, 30
GET /objects/(object_id)/children/(int:child_id),
35
GET /objects/(object_id)/contents, 31
GET /objects/(object_id)/descendants,
32
GET /objects/(object_id)/relations, 33
GET /objects/(object_id)/relations/(string:relation_name),
34
GET /objects/(object_id)/relations/(string:relation_name)/(int:related_id),
34
GET /objects/(object_id)/sections, 31
GET /objects/(object_id)/siblings, 32
POST /objects, 39
POST /objects/(object_id)/children, 42
POST /objects/(object_id)/relations/(string:relation_name),
40
PUT /objects/(object_id)/children/(int:child_id),
43
PUT /objects/(object_id)/relations/(string:relation_name)/(int:related_id),
41
DELETE /objects/(object_id), 44
DELETE /objects/(object_id)/children/(int:child_id),
45
DELETE /objects/(object_id)/relations/(string:relation_name)/(int:related_id),
44

/poster

GET /poster/(object_id), 46

A

Access token, [9](#)

J

JSON Web Token, [9](#)

JWT, [9](#)

R

Refresh token, [9](#)