MediaMonks REST API Spec

This document defines how the REST API for communication between frontend and backend should look and behave.

For more background on the topics and decision making, please refer to the Working Draft document.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Version history

1.0	2015-03-20	Initial version
1.1	2015-04-07	Added _wrapper query parameter for postMessage and other wrappers Added info about Accept-Language header for the locale query parameter Rename since/until to before/after

Table of Contents

1. Request

1.1. Base Url

The base url of the API SHOULD be separated from the other routes, either by using a subdomain like `http://**api**.domain.com/` or a path like `http://www.domain.com/**api**/`.
This way it's easier to set up rules for only the API in a CDN.

1.1.1. Versioning

An API version is OPTIONAL but RECOMMENDED when dealing with larger projects.
The API SHOULD be visible in the url, and placed as high up as possible, e.g. `/api/**v1**/users/123`.
Switching to an higher API version lets the consumer know there are breaking changes. It is RECOMMENDED that the same call in the older API version still works, but not always feasible.

1.1.2. Resources

The path segments of the url SHOULD use **nouns** for resources, e.g. `/users`, `/articles`, `/products`.
The path segments of the url SHOULD NOT use verbs, e.g. `/getUsers`, `/updateArticle`, `/deleteProduct`.
The resource names MUST be **plural**.

It is RECOMMENDED that resource names should be **concrete** in favor of abstract, e.g. `/news`, `/videos` or `/images` instead of `/items`.

Relations SHOULD be visible in the URL, only when they are relevant to the end resource.
`/articles/123/images` where images relate to article
`/layers/12/frames/34/elements/6` where the hierarchical structure is clear, and removing parts of the url reveals the parent resources.
But **not** `/articles/123/authors/34/posts` where the posts of an author don't relate to the article.

Verbs MAY be used for general actions that don't relate to a particular resource, like searching `/search?q=foo+bar` or converting `/convert?from=EUR&to=CNY&amount=100`.

Verbs MAY also be used when the default HTTP Request Methods are not sufficient, e.g. `/users/123/activate`.
Another option could be to treat the action as a sub-resource and use the HTTP verbs.
For example, GitHub's API lets youstar a gist with `PUT /gists/123/star` and unstar with `DELETE /gists/123/star`.

1.2. Query Parameters

Query parameters MUST be used for all actions that produce a different 'view' of the same resource, like the output format, pagination or filtering.

1.2.1. _format

```
?_format=json
?_format=xml
```

The _format query parameter MAY be used as a fallback for Accept header when viewing the response in the browser, because there it's not possible to set the Accept request header.

In all other scenarios the Accept request header MUST be used.

1.2.2. _wrapper

```
?_format=json&_wrapper=postMessage&callback=eventName
```

Thee `_wrapper` query parameter can be used to use a custom wrapper for your data. When a callback parameter is present but no _wrapper parameter, it will use the default *padding* wrapper, resulting in JSONP output for a json format.

`postMessage` is a custom wrapper for cross-domain or cross-protocol hidden-iframe posting in IE8/9 where the response is sent back via the postMessage API. The callback is used as a unique identifier to match the response back to the request.

1.2.3. _method

```
?_method=put
?_method=delete
?_method=patch
```

In case IE8/9 that cannot POST cross-domain or cross-protocol requests, and using the postMessage solution is not an option, a NOT RECOMMENDED option is to do a GET request with all the data, and adding the **`_method`** query parameter to specify the real Request Method.

In normal situations the X-HTTP-Method-Override request header MUST be used.

1.2.4. pagination

There are 2 types of paginations that SHOULD be supported, Offset and Cursor based pagination. A paginated API call MUST implement one of the two types, but MUST NOT implement both, as they each have their own features and limitations.

Based on the following requirements the correct type of pagination should be chosen:

- **Offset**: when a specific page should be returned.
  Cursor base pagination can only return the next or previous page

- **Offset**: when custom sorting should be possible.
  Cursor base pagination is based upon a defined order (e.g. auto-id or date-created). When implementing custom sorting in Cursor based pagination, you first have to fetch the item from the cursor, and use the sort fields for the offset. This requires an extra query plus additional logic.

- **Cursor**: when new items can be inserted at the start of the data-set.
  Offset based pagination will leave gaps or return duplicates if new items are inserted, because the offset will shift.

- **Cursor**: all other scenarios, because it's way faster.

Cursor/Range/Time based pagination

```
?before=2014-11-09T20:28:46+0200&limit=25
?after=2014-11-09T20:28:46+0200&limit=25
```

Time based pagination defines a before or after together with a limit. The limit MUST be defaulted and capped on the backend.

The request MUST contain one of:

- **`before`** (when fetch the previous page, value = a value referencing the first item of the current set, most of the time a timestamp), or
- **`after`** (when fetching the next page, value = a value referencing the last item of the current set, most of the time a timestamp)

The request MAY contain:

- **`limit`** (default = custom in backend)

Offset based pagination

`?offset=0&limit=25`

Offset based pagination defines an offset and a limit. The limit MUST be defaulted and capped on the backend.

The request MAY contain:

- `offset` (default = 0)
- `limit` (default = custom in backend)

1.2.5. locale

```
?locale=en_GB
?locale=nl
?locale=nl_NL
```

When dealing with localized output or different market sites, and the locale/market cannot be derived from the URL itself, the `locale` query parameter MUST be included in the URL to specify the locale of the website.

Both `xx` and `xx_XX` formats SHOULD be supported as values, where `xx` can ONLY be used as a shortcut if the language and country are the same.

Note: a language or locale in API output is mostly used for content, and almost never for formatting. Values like dates or numbers are specified in their raw units (e.g. a unix timestamp or iso-8601 date).
For formatting the `Accept-Language` header MAY be used when the locale parameter only contains a language.

1.2.6. filtering

```
?state=open
?highlight=1
```

When filtering results, query parameters MUST be used to filter fields on a value.

1.2.7. sort

`?sort=priority,-created_at`

When sorting results, the `sort` query parameter MUST be used. The value MUST contain a list of fields to sort on. A minus sign (-) can be used for descending order.

1.2.8. search

`?q=foo+bar`

When searching, the `q` query parameter MUST be used.

1.3. Headers

1.3.1. Content-Type

```
Content-Type: application/json
Content-Type: application/x-www-form-urlencoded
Content-Type: multipart/form-data
```

When sending data to the server, the content type MUST be specified using the `Content-Type` header.
Default content types of `application/json` , `application/x-www-form-urlencoded` and `multipart/form-data` MUST be supported by the server, while other standard and custom content types MAY be used on a per-project basis.

1.3.2. X-Force-Status-Code-200

```
X-Force-Status-Code-200: 1
```

Clients that cannot read the response body when the Status Code is not 2xx MAY set the `X-Force-Status-Code-200` request header.
When the `X-Force-Status-Code-200` is set the response Status Code is always 200, and the real Status Code is placed on the response body.

1.3.3. X-HTTP-Method-Override

```
X-HTTP-Method-Override: PUT
X-HTTP-Method-Override: DELETE
X-HTTP-Method-Override: PATCH
```

In cases where it is not possible to send the desired Request Method (e.g. older browsers that only support GET and POST), using the `X-HTTP-Method-Override` request header MUST be used.

The `X-HTTP-Method-Override` request header MUST NOT be used for changing a GET to something else, only for POST to something else.

1.3.4. Accept

```
Accept: application/json
```

To specify the output format of the response the `Accept` request header MUST be set.

When viewing the response in the browser where it's not possible to set the `Accept` request header, the _format query parameter MAY be used to specify the output format.

1.4. HTTP Method

`GET`
`POST`
`PUT`
`DELETE`
`HEAD`
`PATCH`

The HTTP method MUST be set to the appropriate action.

The following methods MUST be implemented on the server when appropriate for a resource:

- **GET**
  The GET method means retrieve whatever information (in the form of an entity) is identified by the Request-URI.
  This is a safe operation. Every request should give the same response, so the response should be cacheable.

- **POST**
  The POST method is used to request that the server accept the entity enclosed in the request as a new entity of the resource identified by the Request-URI, or by executing an action on that resource.
  This is an unsafe operation. When repeating the request, the overall state is changed.

- **PUT**
  The PUT method requests that the enclosed entity be stored under the supplied Request-URI.
  The fundamental difference between the POST and PUT requests is reflected in the different meaning of the Request-URI.
  This is an unsafe operation. When repeating the request, the overall data should stay the same.

- **DELETE**
  The DELETE method requests that the server delete the resource identified by the Request-URI.
  This is an unsafe operation. When repeating the request, the overall data should stay the same.

The following methods MAY be implemented when useful on a per-project basis:

- **HEAD**
  The HEAD method is identical to GET except that the server MUST NOT return a message-body in the response. The metainformation contained in the HTTP headers in response to a HEAD request SHOULD be identical to the information sent in response to a GET request. This method can be used for obtaining metainformation about the entity implied by the request without transferring the entity-body itself. This method is often used for testing hypertext links for validity, accessibility, and recent modification.

- **PATCH**
  Used when partially updating data (in favor of PUT). This is an unsafe operation. When repeating the request, the overall data should stay the same.

For reference see official W3 HTTP Method Definitions.

1.5. Body

Empty for get/delete, filled for post/put/patch.
Default for application/x-www-form-urlencoded, multipart/form-data
Follow spec for application/json (http://www.w3.org/TR/html-json-forms/)

JSON data

```
{
  "pet": [
     {
        "species": "Dahut",
        "name": "Hypatia"
     },
     {
        "species": "Felis Stultus",
        "name": "Billie"
     }
  ]
}
```

Also sending files in JSON data

```
{
  "file": [
     {
        "type": "text/plain",
        "name": "dahut.txt",
        "body": "REFBQUFBQUFIVVVVVVVVVVVVVVCEhIQo="
     },
     {
        "type": "text/plain",
        "name": "litany.txt",
        "body": "SSBtdXN0IG5vdCBmZWFyLlxuRmVhciBpcyB0aGUgbWluZC1raWxsZXIuCg=="
     }
  ]
}
```

2. Response

2.1. Headers

2.1.1. Status Code

```
HTTP/1.1 200 OK
HTTP/1.1 201 Created
HTTP/1.1 301 Moved Permanently
HTTP/1.1 404 Not Found
```

A response MUST set a correct Status Code representing the action or output of the action/response.

For reference see official W3 HTTP Status Code Definitions.

For clients that cannot read the response body when the Status Code is not 2xx the X-Force-Status-Code-200 request header MAY be used.

2.2. Body

2.2.1. Data

The response data MUST be enveloped in a "data" property. Other (meta)data will be placed next the "data" property in the root of the response.

```
{
  "data": {
    "firstName": "Arjan",
    "lastName": "van Wijk",
    "name": "Narie"
  }
}
```

2.2.2. Error

In case of an error, the error information MUST be enveloped in an "error" property. Other (meta)data will be placed next the "data" property in the root of the response.

An error object has the fields:

- **code**: the error code as a string
- **message**: a human readable error message that might be shown to the user

The code field can be used to handle specific behavior based on the type of error.

```
{
  "error": {
    "code": "error.auth",
    "message": "Some of the aliases you requested do not exist: foobar"
  }
}
```

2.2.3. Validation Error

A validation error is a normal error, but MUST contain a 'fields' key that contains an array of field objects.
A field object has the fields:

- **field**: the identifier of the field that matches the request structure
- **code**: the error code as a string
- **message**: a human readable error message that might be shown to the user

The code field can be used to handle specific behavior based on the type of error (e.g. show extra UI when the provided username already exists), but SHOULD NOT be used to show translated messages. For translated messages the message in the response SHOULD be used.

```
{
 "error": {
   "code": "error.form.validation",
   "message": "Not all fields are filled incorrectly.",

   "fields": [
     {
       "field": "email", // field name
       "code": "error.form.validation.email_exists", // string code
       "message": "This email address is already in use." // human readable message
     },
     {
       "field": "zipcode",
       "code": "error.form.validation.required",
       "message": "This field is required"
     },
     {
       "field": "zipcode",
       "code": "error.form.validation.max_length",
       "message": "This field should not contain more than 6 characters"
     }
   ]
 }
}
```

2.2.4. Status Code

When the `X-Force-Status-Code-200` is set, the actual status code MUST be placed in the body, where the status code in the header MUST be 200.

```
{
  "statusCode": 200,
  "data": {
    "firstName": "Arjan",
    "lastName": "van Wijk",
    "name": "Narie"
  }
}
```

```
{
  "statusCode": 401,
  "error": {
    "code": "error.auth",
    "message": "Some of the aliases you requested do not exist: foobar"
  }
}
```

2.2.5. Pagination

Offset based pagination

Offset based pagination defines an offset and a limit. The limit should be defaulted and capped on the backend.

The response SHOULD contain:

- offset
- limit, the provided limit, capped by the max limit in the backend or the actual results for the page

The response MAY contain:

- total, because this is an expensive operation, only include this when it is required for application logic

```
{
 "pagination": {
   "total": 56, // optionally included when clients wants this information
   "offset": 30, // offset from request
   "limit": 10 // limit from request
 }
}
```

Time/Cursor based pagination

Time based pagination defines a before or after together with a limit. The limit should be defaulted and capped on the backend.

The response SHOULD contain:

- **before** (This is the cursor that points to the start of the page of data that has been returned, can be used for the previous page), or **after** (This is the cursor that points to the end of the page of data that has been returned, can be used for the next page)
- **limit**, the provided limit, capped by the max limit in the backend or the actual results for the page

The response MAY contain:

- **total**, because this is an expensive operation, only include this when it is required for application logic

```
{
 "pagination": {
   "total": 56, // optionally included when clients wants this information
   "before": "2014-11-09T21:28:46+0200", // 'cursor' for the first item in the result
   "after": "2014-11-09T20:45:12+0200", // 'cursor for the last item in the result
   "limit": 10 // limit from request
 }
}
```

3. General

3.1. JSON Naming

Because the default response format is JSON, javascript naming conventions SHOULD be used, so for properties camelCase is used.

For other conventions, see the Google JSON Style Guide.
https://google-styleguide.googlecode.com/svn/trunk/jsoncstyleguide.xml

Additions to the style guide:

- birthdates as YYYY-MM-DD
- birthdays as MM-DD