OAR Documentation - REST API - User's doc

Dedication: For users whishing to make programs interfaced with OAR

Abstract: OAR is a resource manager (or batch scheduler) for large clusters. By it's functionnalities, it's near of PBS, LSF, CCS and Condor. It's suitable for productive plateforms and research experiments.

BE CAREFULL : THIS DOCUMENTATION IS FOR OAR >= 2.5.0
PDF version : OAR-DOCUMENTATION-API-USER.pdf

Table of Contents

Introduction

The OAR REST API allows to interact with OAR over http using a REST library. Most of the operations usually done with the oar Unix commands may be done using this API from your favourite language.

Concepts

Access

A simple GET query to the API using wget may look like this:

```
# Get the list of resources
wget -O - http://www.mydomain.org/oarapi/resources.yaml?structure=simple
```

You can also access to the API using a browser. Make it point to http://www.myoarcluster.local/oarapi/in and you'll see a very simple HTML interface allowing you to browse the cluster resources, post a job using a form or even create resources if you are a OAR administrator. (of course, replace www.myoarcluster.local by a valid name allowing you to join the http service of the host where the API is installed).

But generally, you'll use a REST client or a REST library provided for your favorite language. You'll see examples using a ruby rest library in the next parts of this document.

Check your system administrator to know on which URI the OAR API is installed.

Authentication

Most of the time, you'll make requests that needs you to be authenticated. The way you are authenticated depends on what your local admistrator configured. There's almost as many possibilities as what Apache (the http server used by this API) may manage. The simplest method is a "Basic authentication" with a login/password. It may be binded to a local directory (for example LDAP). You may also find an "ident" based authentication that guesses automatically your login from a little daemon running on your client host. If the "ident" method is used, your unix login is automatically used. But as only a few hosts may be trusted, you'll probably have to open a tunnel to one of this host. You may use ssh to do this. For example, supposing access.mycluster.fr is a gateway host trusted by the api host:

```
$ ssh -NL 8080:api.mycluster.fr:80 login@access.mycluster.fr

Then, point your REST client to::

# http://localhost:8080
```

Formats and data structure types

The API currently can serve data into *YAML*, *JSON* or *HTML*. Posted data can also be coded into *YAML*, *JSON* or *x-www-form-urlencoded* (for HTML from posts). You may specify the requested format by 2 ways:

- *giving an extension to resources:* ***.yaml***, ***.json*** *or* ***.html***
- *setting the* ***HTTP_ACCEPT*** *header variable to* ***text/yaml***, ***application/json*** *or* ***text/html***

For the posted data, you have to correctly set the **HTTP_CONTENT_TYPE** variable to **text/yaml**, **application/json** or **application/x-www-form-urlencoded**.

Sometimes, the data structures returned (not the coding format, but the contents: array, hashes, array of hashes,...) may be changed. Currently, we have 2 available data structure types: *simple* and *oar*. The structure is passed through the variable *structure* that you may pass in the url, for example: ?structure=simple

- *The **simple** data structure tries to be as simple as possible, using simple arrays in place of hashes wherever it is possible*
- *The **oar** data structure serves data in the way oar does with the oarnodes/oarstat export options (-Y, -D, -J,...) Be aware that this data structure is not meant to be maintained since 2.5 release of OAR. The simple data structure is highly recommended.*

By default, we use the *simple* data structure.

Here are some examples, using the ruby restclient (see next section):

```
# Getting resources infos
  # in JSON
irb(main):004:0> puts get('/resources.json')
  # in YAML
irb(main):005:0> puts get('/resources.yaml')
  # Same thing
irb(main):050:0> puts get('/resources', :accept=>"text/yaml")
  # Specifying the "oar" data structure
irb(main):050:0> puts get('/resources.json?structure=oar')
  # Specifying the "simple" data structure
irb(main):050:0> puts get('/resources.json?structure=simple')
```

Errors and debug

When the API returns an error, it generally uses a standard HTTP return status (404 NOT FOUND, 406 NOT ACCEPTABLE, ...). But it also returns a body containing a hash like the following:

```
{
 "title" : "ERROR 406 - Invalid content type required */*",
 "message" : "Valid types are text/yaml, application/json or text/html",
 "code" : "200"
}
```

This error body is formated in the requested format. But if this format was not given, it uses JSON by default.

To allow you to see the error body, you may find it useful to activate the **debug=1** variable. It will force the API to always return a 200 OK status, even if there's an error

so that you can see the body with a simple browser or a rest client without having to manage the errors. For example:

```
wget -nv -O - "http://localhost:8080/oargridapi/sites/grenoble?debug=1"
```

Here is an example of error catching in ruby:

```
# Function to get objects from the api
# We use the JSON format
def get(api,uri)
  begin
    return JSON.parse(api[uri].get(:accept => 'application/json'))
  rescue => e
    if e.respond_to?('http_code')
      puts "ERROR #{e.http_code}:\n #{e.response.body}"
    else
      puts "Parse error:"
      puts e.inspect
    end
    exit 1
  end
end
```

Ruby REST client

One of the easiest way for testing this API is to use the rest-client ruby module: http://rest-client.heroku.com/rdoc/

It may be used from ruby scripts (http://www.ruby.org/) or interactively. It is available as a rubygem, so to install it, simply install rubygems and do "gem install rest-client". Then, you can run the interactive client which is nothing else than irb with shortcuts. Here is an example irb session:

```
$ export PATH=$PATH:/var/lib/gems/1.8/bin
$ restclient http://localhost/oarapi
irb(main):001:0> puts get('/jobs.yaml')
---
- api_timestamp: 1246457384
  id: 551
  name: ~
  owner: bzizou
  queue: besteffort
  state: Waiting
  submission: 1245858042
  uri: /jobs/551
=> nil
```

```
irb(main):002:0>
```

or, if an http basic auth is required:

```
restclient http://localhost/api <login> <password>
...
```

Pagination and common rules into output data

Results served by the API are mainly of 2 kinds: “items” and “collections”. A collection is actually an array of items. Some uris serve collections that may have a very big amount of items (for example all the terminated jobs of a cluster). For that reason, collections are often “paginated”. It means that the collections are presented into pages that have got meta data to give you informations about offset, numbers, and links to previous/next page. Furthermore, items are often composed of commonly used kind of data, especially ’id’ and ’links’. We have tried to normalize this as much as possible, so, here is a description of the common properties of items and collections:

Items

Items have the following features:

Hash: Items should be hashes (sometimes hash of hashes for the ’oar’ data structure, but it is to be avoided)

the ’id’ key: In general, when an item may be uniquely identified by an integer, it is given in the “id” key. Note that OAR nodes are identified by the ’network_address’ key that is an integer, but this is an exception.

the ’links’ array:
Items, especially when listed in a collection, often give links to more informations or relative data. The links are listed in the links array. Each element of this array (a link) is composed of at least: a ’rel’ key and a ’href’ key. The ’rel’ key is a string telling what is the relation between the current item and the resource pointed by the link. The ’href’ key is a string giving the URI of the link relative to the root of the API. It’s possible that other keys will be implemented in the future (for example a ’title’ key.) Common values for ’rel’ are: ’self’, ’parent’, ’next’, ’previous’.

the ’api_timestamp’ value:
Each item has a ’api_timestamp’ key giving the epoch unix date at which the API constructed the item. This field may be omitted when items are listed inside a collection; then the collection has a global api_timestamp value. This date is given in the timezone provided by the “GET /timezone uri”.

Collections

Collections have the following features:

the 'items' array:
: The items array is the purpose of a collection. It lists all the items of the current page of a collection.

the 'total' number:
: It's an integer giving the total number of items in the collection. If the items array contains less elements than this number, then the collection has been paginated and a 'next' and/or 'previous' link should be provided.

the 'offset' number:
: It gives the offset at which the paginated list starts. If 0, then, it is the first page.

the 'limit' parameter:
: This is not in the output, but a parameter common to all paginable uris. If you specify a limit, then it gives the size of the pages.

the 'links' array:
: For a collection, the links array often gives the uri of the next/previous page. But it also gives the uri of the current page ('self') and may point to more informations relative to this collection. See the links array description from above for items, it is similar for the collection.

Examples

An item looks like this (yaml output):

```
api_timestamp: 1286894740
available_upto: 2147483646
besteffort: YES
core: 1
cpu: 1
cpuset: 0
deploy: NO
desktop_computing: NO
expiry_date: 0
finaud_decision: NO
id: 1
last_available_upto: 0
last_job_date: 1286885902
links:
  - href: /resources/nodes/fake1
    rel: node
  - href: /resources/1
    rel: self
  - href: /resources/1/jobs
```

```
    rel: jobs
network_address: fake1
next_finaud_decision: NO
next_state: UnChanged
resource_id: 1
scheduler_priority: 0
state: Alive
state_num: 1
suspended_jobs: NO
type: default
```

A collection looks like this (yaml output):

```
api_timestamp: 1286894823
items:
  - api_timestamp: 1286894823
    id: 2
    links:
      - href: /jobs/2
        rel: self
      - href: /jobs/2/resources
        rel: resources
    name: ~
    owner: kameleon
    queue: default
    state: Error
    submission: 1284034267
  - api_timestamp: 1286894823
    id: 3
    links:
      - href: /jobs/3
        rel: self
      - href: /jobs/3/resources
        rel: resources
    name: ~
    owner: kameleon
    queue: default
    state: Error
    submission: 1284034383
links:
  - href: /jobs.yaml?state=Error&limit=2&offset=0
    rel: self
  - href: /jobs.yaml?state=Error&limit=2&offset=2
    rel: next
offset: 0
```

```
total: 2623
```

REST requests description

Examples are given in the YAML format because we think that it is the more human readable and so very suitable for this kind of documentation. But you can also use the JSON format for your input/output data. Each resource uri may be postfixed by .yaml, .jso of .html.

In this section, we describe every REST resources of the OAR API. The authentication may be:

- public: everybody can query this resource
- user: only authenticated and valid users can query this resource
- oar: only the oar user can query this resource (administration usage)

GET /index

description: Home page for the HTML browsing
formats: html
authentication:
public
output: *example*:

```
<HTML>
<HEAD>
<TITLE>OAR REST API</TITLE>
</HEAD>
<BODY>
<HR>
<A HREF=./resources.html>RESOURCES</A>   
<A HREF=./jobs.html>JOBS</A>   
<A HREF=./jobs/form.html>SUBMISSION</A>   
<HR>
Welcome on the oar API
```

note: Header of the HTML resources may be customized into the **/etc/oar/api_html_header.pl** file.

GET /version

description: Gives version informations about OAR and OAR API. Also gives the timezone of the API server.
formats: html , yaml , json
authentication:
public

output: *structure*:
hash
yaml example:

```
---
api: 0.1.2
api_timestamp: 1245582255
api_timezone: CEST
apilib: 0.1.6
oar: 2.4.0
```

usage example:

```
wget -q -O - http://localhost/oarapi/version.yaml
```

GET /timezone

description: Gives the timezone of the OAR API server. The api_timestamp given in each query is an UTC timestamp (epoch unix time). This timezone information allows you to re-construct the local time.
formats: html , yaml , json
authentication:
public
output: *structure*: hash
yaml example:

```
---
api_timestamp: 1245768107
timezone: CEST
```

usage example:

```
wget -q -O - http://localhost/oarapi/timezone.yaml
```

GET /jobs

description: List jobs (by default only the jobs in queue)
formats: html , yaml , json
authentication:
public
parameters:

- **state**: comma separated list of states for filtering the jobs. Possible values: Terminated, Running, Error, Waiting, Launching, Hold,...
- **array** (integer): to get the jobs belonging to an array
- **from** (timestamp): restrict the list to the jobs that are running or not yet started before this date. Using this parameters disables the default behavior of listing only the jobs that are in queue.

- **to** (timestamp): restrict the list to the jobs that are running or not yet finished at this date. Using this parameters disables the default behavior of listing only the jobs that are in queue.
- **user**: restrict the list to the jobs owned by this username

output: *structure*: collection

yaml example:

```
api_timestamp: 1286895857
items:
  - api_timestamp: 1286895857
    id: 58
    links:
      - href: /jobs/58
        rel: self
      - href: /jobs/58/resources
        rel: collection
        title: resources
      - href: /oarapi/jobs/58/nodes
        rel: collection
        title: nodes
    name: ~
    owner: kameleon
    queue: default
    state: Terminated
    submission: 1284109267
  - api_timestamp: 1286895857
    id: 59
    links:
      - href: /jobs/59
        rel: self
      - href: /jobs/59/resources
        rel: collection
        title: resources
      - href: /oarapi/jobs/59/nodes
        rel: collection
        title: nodes
    name: ~
    owner: kameleon
    queue: default
    state: Terminated
    submission: 1284109846
links:
  - href: /jobs.yaml?state=Terminated&limit=2&offset=48
    rel: self
```

```
    - href: /jobs.yaml?state=Terminated&limit=2&offset=50
      rel: next
    - href: /jobs.yaml?state=Terminated&limit=2&offset=46
      rel: previous
  offset: 48
  total: 206
```

note: The "rel: resources" link of a job lists the assigned or reserved resources of a job.

usage example:

```
wget -q -O - http://localhost/oarapi/jobs.yaml?state=Terminated,Running&l
```

GET /jobs/details

description: Same as /jobs, but with more details and "resources" and "nodes" links developped.

formats: html , yaml , json

authentication:

public

parameters:

- **state**: comma separated list of states for filtering the jobs. Possible values: Terminated, Running, Error, Waiting, Launching, Hold,...

output: *structure*: collection

yaml example:

```
api_timestamp: 1352707511
items:
  - api_timestamp: 1352707511
    array_id: 5540
    array_index: ~
    command: sleep 300
    cpuset_name: kameleon_5540
    dependencies: []
    events: []
    exit_code: ~
    id: 5540
    initial_request: oarsub sleep 300
    launching_directory: /home/kameleon
    links:
      - href: /oarapi/jobs/5540
        rel: self
      - href: /oarapi/jobs/5540/resources
        rel: collection
        title: resources
      - href: /oarapi/jobs/5540/nodes
        rel: collection
```

```
    title: nodes
  message: Karma = 0.000
  name: ~
  nodes:
    - api_timestamp: 1352707511
      links:
        - href: /oarapi/resources/nodes/node1
          rel: self
      network_address: node1
      status: assigned
  owner: kameleon
  project: default
  properties: desktop_computing = 'NO'
  queue: default
  reservation: None
  resources:
    - api_timestamp: 1352707511
      id: 1
      links:
        - href: /oarapi/resources/1
          rel: self
        - href: /oarapi/resources/1/jobs
          rel: collection
          title: jobs
      status: assigned
  resubmit_job_id: ~
  scheduled_start: 1352707488
  start_time: 1352707488
  state: Running
  stderr_file: OAR.5540.stdout
  stdout_file: OAR.5540.stderr
  stop_time: 0
  submission_time: 1352707487
  type: PASSIVE
  types: []
  walltime: 7200
  wanted_resources: "-l \"{type = 'default'}/resource_id=1,walltime
- api_timestamp: 1352707511
  array_id: 5542
  array_index: ~
  command: sleep 300
  cpuset_name: kameleon_5542
  dependencies: []
  events: []
```

```
exit_code: ~
id: 5542
initial_request: oarsub -l /core=2 sleep 300
launching_directory: /home/kameleon
links:
  - href: /oarapi/jobs/5542
    rel: self
  - href: /oarapi/jobs/5542/resources
    rel: collection
    title: resources
  - href: /oarapi/jobs/5542/nodes
    rel: collection
    title: nodes
message: Karma = 0.000
name: ~
nodes:
  - api_timestamp: 1352707511
    links:
      - href: /oarapi/resources/nodes/node1
        rel: self
    network_address: node1
    status: assigned
owner: kameleon
project: default
properties: desktop_computing = 'NO'
queue: default
reservation: None
resources:
  - api_timestamp: 1352707511
    id: 3
    links:
      - href: /oarapi/resources/3
        rel: self
      - href: /oarapi/resources/3/jobs
        rel: collection
        title: jobs
    status: assigned
  - api_timestamp: 1352707511
    id: 4
    links:
      - href: /oarapi/resources/4
        rel: self
      - href: /oarapi/resources/4/jobs
        rel: collection
```

```
            title: jobs
          status: assigned
        resubmit_job_id: ~
        scheduled_start: 1352707510
        start_time: 1352707510
        state: Running
        stderr_file: OAR.5542.stdout
        stdout_file: OAR.5542.stderr
        stop_time: 0
        submission_time: 1352707509
        type: PASSIVE
        types: []
        walltime: 7200
        wanted_resources: "-l \"{type = 'default'}/core=2,walltime=2:0:0\
    links:
      - href: /oarapi/jobs/details.yaml?offset=0
        rel: self
    offset: 0
    total: 2
```

usage example:

```
wget -q -O - http://localhost/oarapi/jobs/details.yaml
```

GET /jobs/table

description: Same as /jobs but outputs the data of the SQL job table
formats: html , yaml , json
authentication: public
parameters:
- **state**: comma separated list of states for filtering the jobs. Possible values: Terminated, Running, Error, Waiting, Launching, Hold,...

output: *structure*: collection

yaml example:

```
---
items:
 - accounted: NO
   api_timestamp: 1253017554
   array_id: 566
   assigned_moldable_job: 566
   checkpoint: 0
   checkpoint_signal: 12
   command: ''
```

```
  exit_code: ~
  file_id: ~
  info_type: bart:33033
  initial_request: oarsub -I
  job_env: ~
  job_group: ''
  job_id: 566
  job_name: ~
  job_type: INTERACTIVE
  job_user: bzizou
  launching_directory: /home/bzizou/git/oar/git
  message: FIFO scheduling OK
  notify: ~
  project: default
  properties: desktop_computing = 'NO'
  queue_name: default
  reservation: None
  resubmit_job_id: 0
  scheduler_info: FIFO scheduling OK
  start_time: 1253017553
  state: Launching
  stderr_file: OAR.%jobid%.stderr
  stdout_file: OAR.%jobid%.stdout
  stop_time: 0
  submission_time: 1253017551
  suspended: NO
  uri: /jobs/566
- accounted: NO
  api_timestamp: 1253017554
  array_id: 560
  assigned_moldable_job: 0
  checkpoint: 0
  checkpoint_signal: 12
  command: /usr/bin/id
  exit_code: ~
  file_id: ~
  info_type: 'bart:'
  initial_request: oarsub --resource=/nodes=2/cpu=1 --use_job_key=1
  job_env: ~
  job_group: ''
  job_id: 560
  job_name: ~
  job_type: PASSIVE
  job_user: bzizou
```

```
    launching_directory: /home/bzizou
    message: Cannot find enough resources which fit for the job 560
    notify: ~
    project: default
    properties: desktop_computing = 'NO'
    queue_name: default
    reservation: None
    resubmit_job_id: 0
    scheduler_info: Cannot find enough resources which fit for the job
    start_time: 0
    state: Waiting
    stderr_file: OAR.%jobid%.stderr
    stdout_file: OAR.%jobid%.stdout
    stop_time: 0
    submission_time: 1246948570
    suspended: NO
    uri: /jobs/560
links:
 - href: '/jobs/table.html?state=Terminated&limit=15&offset=0'
   rel: previous
 - href: '/jobs/table.html?state=Terminated&limit=15&offset=15'
   rel: self
 - href: '/jobs/table.html?state=Terminated&limit=15&offset=30'
   rel: next
offset: 15
total: 41
```

note: Field names may vary from the other job lists because this query results more like a dump of the jobs table.

usage example:

```
wget -q -O - http://localhost/oarapi/jobs/table.yaml
```

GET /jobs/<id>

description: Get details about the given job

parameters:
- **id**: the id of a job

formats: html , yaml , json

authentication: user

output: *structure*: hash

yaml example:

```
api_timestamp: 1352707658
array_id: 5230
array_index: 3
```

```
command: /home/kameleon/cigri-3/tmp/test1.sh param48 48
cpuset_name: kameleon_5232
dependencies: []
events:
  - date: 1351087783
    description: Scheduler priority for job 5232 updated (network_adc
    event_id: 14454
    job_id: 5232
    to_check: NO
    type: SCHEDULER_PRIORITY_UPDATED_STOP
  - date: 1351087782
    description: '[bipbip 5232] Ask to change the job state'
    event_id: 14451
    job_id: 5232
    to_check: NO
    type: SWITCH_INTO_TERMINATE_STATE
  - date: 1351087660
    description: Scheduler priority for job 5232 updated (network_adc
    event_id: 14446
    job_id: 5232
    to_check: NO
    type: SCHEDULER_PRIORITY_UPDATED_START
exit_code: 0
id: 5232
initial_request: oarsub --resource=core=1 --type=besteffort /home/kam
launching_directory: /home/kameleon
links:
  - href: /oarapi/jobs/5232
    rel: self
  - href: /oarapi/jobs/5232/resources
    rel: collection
    title: resources
  - href: /oarapi/jobs/5232/nodes
    rel: collection
    title: nodes
message: Karma = 0.000
name: ~
owner: kameleon
project: default
properties: (besteffort = 'YES') AND desktop_computing = 'NO'
queue: besteffort
reservation: None
resubmit_job_id: 0
scheduled_start: ~
```

```
    start_time: 1351087660
    state: Terminated
    stderr_file: OAR.5232.stderr
    stdout_file: OAR.5232.stdout
    stop_time: 1351087782
    submission_time: 1351087659
    type: PASSIVE
    types:
      - besteffort
    walltime: 7200
    wanted_resources: "-l \"{type = 'default'}/core=1,walltime=2:0:0\" "
```

usage example:

```
wget --user test --password test -q -O - http://localhost/oarapi/jobs/547
```

GET /jobs/<id>/resources

description: Get resources reserved or assigned to a job
parameters:
- **id**: the id of a job

formats: html , yaml , json
authentication:
public
output: *structure*: hash

yaml example:

```
api_timestamp: 1352707730
items:
  - api_timestamp: 1352707730
    id: 7
    links:
      - href: /oarapi/resources/7
        rel: self
      - href: /oarapi/resources/7/jobs
        rel: collection
        title: jobs
    status: assigned
links:
  - href: /oarapi/jobs/5232/resources.yaml
    rel: self
offset: 0
total: 1
```

usage example:

```
wget -q -O - http://localhost/oarapi/jobs/547/resources.yaml
```

POST /jobs/<id>/deletions/new

description: Deletes a job
parameters:
- **id**: the id of a job

formats: html , yaml , json
authentication: user
output: *structure*: hash

yaml example:

```
---
api_timestamp: 1253025331
cmd_output: |
  Deleting the job = 567 ...REGISTERED.
  The job(s) [ 567 ] will be deleted in a near future.
id: 567
status: Delete request registered
```

usage example:

```
irb(main):148:0> puts post('/jobs/567/deletions/new.yaml','')
```

POST /jobs/<id>/checkpoints/new

description: Send the checkpoint signal to a job
parameters:
- **id**: the id of a job

formats: html , yaml , json
authentication: user
output: *structure*: hash

yaml example:

```
---
api_timestamp: 1253025555
cmd_output: |
  Checkpointing the job 568 ...DONE.
  The job 568 was notified to checkpoint itself.
id: 568
status: Checkpoint request registered
```

usage example:

```
irb(main):148:0> puts post('/jobs/568/checkpoints/new.yaml','')
```

POST /jobs/<id>/holds/new

description: Asks to hold a waiting job
parameters:
- **id**: the id of a job

formats: html , yaml , json

authentication:
user
output: *structure*: hash

yaml example:

```
---
api_timestamp: 1253025718
cmd_output: "[560] Hold request was sent to the OAR server.\n"
id: 560
status: Hold request registered
```

usage example:

```
irb(main):148:0> puts post('/jobs/560/holds/new.yaml','')
```

POST /jobs/<id>/rholds/new

description: Asks to hold a running job
parameters:
- **id**: the id of a job

formats: html , yaml , json
authentication:
oar
output: *structure*: hash

yaml example:

```
---
api_timestamp: 1253025868
cmd_output: "[569] Hold request was sent to the OAR server.\n"
id: 569
status: Hold request registered
```

usage example:

```
irb(main):148:0> puts post('/jobs/560/rholds/new.yaml','')
```

POST /jobs/<id>/resumptions/new

description: Asks to resume a holded job
parameters:
- **id**: the id of a job

formats: html , yaml , json
authentication:
user
output: *structure*: hash

yaml example:

```
---
api_timestamp: 1253026081
cmd_output: "[569] Resume request was sent to the OAR server.\n"
id: 569
status: Resume request registered
```

usage example:

```
irb(main):148:0> puts post('/jobs/560/resumptions/new.yaml','')
```

POST /jobs/<id>/signals/<signal>

description: Asks to resume a holded job

parameters:

- **id**: the id of a job
- **signal**: the number of a signal (see kill -l)

formats: html , yaml , json

authentication: user

output: *structure*: hash

yaml example:

```
---
api_timestamp: 1253102493
cmd_output: |
  Signaling the job 574 with 12 signal.
  DONE.
  The job 574 was notified to signal itself with 12.
id: 574
status: Signal sending request registered
```

usage example:

```
irb(main):148:0> puts post('/jobs/560/signals/12.yaml','')
```

POST /jobs

description: Creates (submit) a new job

formats: html , yaml , json

authentication: user

input: Only [resource] and [command] are mandatory. Please, refer to the documentation of the *oarsub* command for the resource syntax which correspond to the -l (--resource) option.

structure: hash with possible arrays (for options that may be passed multiple times)

fields:

- **resource** (*string*): the resources description as required by oar (example: "/nodes=1/cpu=2")
- **command** (*string*): a command name or a script that is executed when the job starts
- **workdir** (*string*): the path of the directory from where the job will be submited

- **param_file** (*string*): the content of a parameters file, for the submission of an array job. For example: {“resource”:“/nodes=1, ”command“:”sleep“, ”param_file“:”60n90n30“}
- **All other option accepted by the oarsub unix command**: every long option that may be passed to the oarsub command is known as a key of the input hash. If the option is a toggle (no value), you just have to set it to “1” (for example: ’use-job-key’ => ’1’). Some options may be arrays (for example if you want to specify several ’types’ for a job)

yaml example:

```
---
stdout: /tmp/outfile
command: /usr/bin/id;echo "OK"
resource: /nodes=2/cpu=1
workdir: ~bzizou/tmp
type:
- besteffort
- timesharing
use-job-key: 1
```

output: *structure*: hash

yaml example:

```
---
api_timestamp: 1332323792
cmd_output: |
  [ADMISSION RULE] Modify resource description with type constraints
  OAR_JOB_ID=4
id: 4
links:
  - href: /oarapi-priv/jobs/4
    rel: self
```

note: more informations about the submited job may be obtained with a GET on the provided *uri*.

usage example:

```
# Submitting a job using ruby rest client
irb(main):010:0> require 'json'
irb(main):012:0> j={ 'resource' => '/nodes=2/cpu=1', 'command' => '/usr/b
irb(main):015:0> job=post('/jobs' , j.to_json , :content_type => 'applica

# Submitting a job with a provided inline script
irb(main):024:0> script="#!/bin/bash
irb(main):025:0" echo \"Hello world\"
irb(main):026:0" whoami
irb(main):027:0" sleep 300
```

```
irb(main):028:0" "
irb(main):029:0> j={ 'resource' => '/nodes=2/cpu=1', 'script' => script ,
irb(main):030:0> job=post('/jobs' , j.to_json , :content_type => 'applica
```

POST /jobs/<id>

description: Updates a job. In fact, as some clients (www browsers) doesn't support the DELETE method, this POST resource has been created mainly to workaround this and provide another way to delete a job. It also provides *checkpoint*, *hold* and *resume* methods, but one should preferably use the /checkpoints, /holds and /resumptions resources.

formats: html , yaml , json

authentication:
user

input: *structure*: hash {"action" => "delete"}

yaml example:

```
---
method: delete
```

output: *structure*: hash

yaml example:

```
---
api_timestamp: 1245944206
cmd_output: |
  Deleting the job = 554 ...REGISTERED.
  The job(s) [ 554 ] will be deleted in a near future.
id: 554
status: Delete request registered
```

usage example:

```
# Deleting a job in the ruby rest client
puts post('/jobs/554.yaml','{"method":"delete"}',:content_type => "applic
```

DELETE /jobs/<id>

description: Delete or kill a job.

formats: html , yaml , json

authentication:
user

output: *structure*: hash returning the status

yaml example:

```
---
api_timestamp: 1245944206
cmd_output: |
  Deleting the job = 554 ...REGISTERED.
```

```
  The job(s) [ 554 ] will be deleted in a near future.
id: 554
status: Delete request registered
```

usage example:

```
# Deleting a job in the ruby rest client
puts delete('/jobs/554.yaml')
```

note: Not all clients support the DELETE method, especially some www browsers. So, you can do the same thing with a POST of a {"method":"delete"} hash on the /jobs/<id> resource.

GET /jobs/form

description: HTML form for posting (submiting) new jobs from a browser
formats: html
authentication: user
output: *example*:

```
<HTML>
 <HEAD>
 <TITLE>OAR REST API</TITLE>
 </HEAD>
 <BODY>
 <HR>
 <A HREF=../resources.html>RESOURCES</A>   
 <A HREF=../jobs.html>JOBS</A>   
 <A HREF=../jobs/form.html>SUBMISSION</A>   
 <HR>

 <FORM METHOD=post ACTION=../jobs.html>
 <TABLE>
 <CAPTION>Job submission</CAPTION>
 <TR>
   <TD>Resources</TD>
   <TD><INPUT TYPE=text SIZE=40 NAME=resource VALUE="/nodes=1/cpu=1,w
 </TR><TR>
   <TD>Name</TD>
   <TD><INPUT TYPE=text SIZE=40 NAME=name VALUE="Test_job"></TD>
 </TR><TR>
   <TD>Properties</TD>
   <TD><INPUT TYPE=text SIZE=40 NAME=property VALUE=""></TD>
 </TR><TR>
   <TD>Program to run</TD>
   <TD><INPUT TYPE=text SIZE=40 NAME=command VALUE='"/bin/sleep 300"
 </TR><TR>
```

```
  <TD>Types</TD>
  <TD><INPUT TYPE=text SIZE=40 NAME=type></TD>
</TR><TR>
  <TD>Reservation dates</TD>
  <TD><INPUT TYPE=text SIZE=40 NAME=reservation></TD>
</TR><TR>
  <TD>Directory</TD>
  <TD><INPUT TYPE=text SIZE=40 NAME=directory></TD>
</TR><TR>
  <TD></TD><TD><INPUT TYPE=submit VALUE=SUBMIT></TD>
</TR>
</TABLE>
</FORM>
```

note: This form may be customized in the **/etc/oar/api_html_postform.pl** file

GET /resources

description: Get the list of resources and their state
formats: html , yaml , json
authentication:
public
output: *structure*: hash

fields:

- **items** : list of resources
- **links** : links to previous, current and next resources
- **offset** : current offset
- **total** : resources total

yaml example:

```
---
items:
 - api_timestamp: 1253201950
   jobs_uri: /resources/4/jobs
   network_address: liza-1
   node_uri: /resources/nodes/liza-1
   resource_id: 4
   state: Alive
   uri: /resources/4
 - api_timestamp: 1253201950
   jobs_uri: /resources/5/jobs
   network_address: liza-1
   node_uri: /resources/nodes/liza-1
   resource_id: 5
   state: Alive
```

```
    uri: /resources/5
  - api_timestamp: 1253201950
    jobs_uri: /resources/6/jobs
    network_address: liza-2
    node_uri: /resources/nodes/liza-2
    resource_id: 6
    state: Alive
    uri: /resources/6
  - api_timestamp: 1253201950
    jobs_uri: /resources/7/jobs
    network_address: liza-2
    node_uri: /resources/nodes/liza-2
    resource_id: 7
    state: Alive
    uri: /resources/7
 links:
  - href: '/resources.yaml?limit=5&offset=2'
    rel: previous
  - href: '/resources.yaml?limit=5&offset=7'
    rel: self
  - href: '/resources.yaml?limit=5&offset=12'
    rel: next
 offset: 2
 total: 49
```


note: More details about a resource can be obtained with a GET on the provided *uri*. The list of all the resources of the same node may be obtained with a GET on *node_uri*. The list of running jobs on a resource can be obtained with a GET on the jobs_uri resource. *note*: The following parameters can be passed through the requested URL

- *limit : limit of resources to be shown per page*
- *offset : the page result offset*

usage example:

```
wget -q -O - http://localhost/oarapi/resources.yaml
```

GET /resources/full

description: Get the list of resources and all the details about them
formats: html , yaml , json
authentication:
public
output: *structure*: hash

fields:

- **items** : list of resources

- **links** : links to previous, current and next resources
- **offset** : current offset
- **total** : total of resources

yaml example:

```
---
    items:
          - api_timestamp: 1281967035
     available_upto: 0
     besteffort: YES
     core: ~
     cpu: 0
     cpufreq: ~
     cpuset: 0
     cputype: ~
     deploy: NO
     desktop_computing: NO
     expiry_date: 0
     finaud_decision: NO
     jobs_uri: '/resources/1/jobs.html'
     last_available_upto: 0
     last_job_date: 1278588052
     memnode: ~
     network_address: node1
         next_finaud_decision: NO
         next_state: UnChanged
         node_uri: '/resources/nodes/node1.html'
         resource_id: 1
         scheduler_priority: 0
         state: Suspected
         state_num: 3
         suspended_jobs: NO
         type: default
         uri: '/resources/1.html'
           - api_timestamp: 1281967035
         available_upto: 0
         besteffort: YES
         core: ~
         cpu: 0
         cpufreq: ~
         cpuset: 0
         cputype: ~
         deploy: NO
         desktop_computing: NO
         expiry_date: 0
```

```
    finaud_decision: NO
    jobs_uri: '/resources/2/jobs.html'
    last_available_upto: 0
    last_job_date: 1278588052
    memnode: ~
    network_address: node1
    next_finaud_decision: NO
    next_state: UnChanged
    node_uri: '/resources/nodes/node1.html'
    resource_id: 2
    scheduler_priority: 0
    state: Suspected
    state_num: 3
    suspended_jobs: NO
    type: default
    uri: '/resources/2.html'
      - api_timestamp: 1281967035
    available_upto: 0
    besteffort: YES
    core: ~
    cpu: 1
    cpufreq: ~
    cpuset: 0
    cputype: ~
    deploy: NO
    desktop_computing: NO
    expiry_date: 0
    finaud_decision: NO
    jobs_uri: '/resources/3/jobs.html'
    last_available_upto: 0
    last_job_date: 1278588052
    memnode: ~
    network_address: node1
    next_finaud_decision: NO
    next_state: UnChanged
    node_uri: '/resources/nodes/node1.html'
    resource_id: 3
    scheduler_priority: 0
    state: Suspected
    state_num: 3
    suspended_jobs: NO
    type: default
    uri: '/resources/3.html'
links:
```

```
                        - href: '/resources/full.yaml?limit=5&offset=2'
                          rel: previous
                        - href: '/resources/full.yaml?limit=5&offset=7'
                          rel: self
                        - href: '/resources/full.yaml?limit=5&offset=12'
                          rel: next
                    offset: 2
                        total: 49
```

usage example:

```
wget -q -O - http://localhost/oarapi/resources/full.yaml

*note*: The following parameters can be passed through the requested URL
       - limit : limit of resources to be shown per page
       - offset : the page result offset
```

GET /resources/<id>

description: Get details about the resource identified by *id*
formats: html , yaml , json
authentication:
public
output: *structure*: 1 element array of hash

yaml example:

```
---
api_timestamp: 1253202322
available_upto: 0
besteffort: YES
cluster: 0
cpu: 20
cpuset: 0
deploy: NO
desktop_computing: NO
expiry_date: 0
finaud_decision: NO
jobs_uri: /resources/1/jobs
last_available_upto: 0
last_job_date: 1253201845
licence: ~
network_address: bart-1
next_finaud_decision: NO
next_state: UnChanged
node_uri: /resources/nodes/bart-1
resource_id: 1
scheduler_priority: 0
```

```
    state: Alive
    state_num: 1
    suspended_jobs: NO
    test: ~
    type: default
    uri: /resources/1
```

usage example:

```
wget -q -O - http://localhost/oarapi/resources/1.yaml
```

GET /resources/nodes/<network_address>

description: Get details about the resources belonging to the node identified by *network_address*

formats: html , yaml , json

authentication:

public

output: *structure*: array of hashes

yaml example:

```
---
- api_timestamp: 1253202379
  jobs_uri: /resources/4/jobs
  network_address: liza-1
  node_uri: /resources/nodes/liza-1
  resource_id: 4
  state: Alive
  uri: /resources/4
- api_timestamp: 1253202379
  jobs_uri: /resources/5/jobs
  network_address: liza-1
  node_uri: /resources/nodes/liza-1
  resource_id: 5
  state: Alive
  uri: /resources/5
```

usage example:

```
wget -q -O - http://localhost/oarapi/resources/nodes/liza-1.yaml
```

POST /resources/generate

description: Generates (outputs) a set of resources using oaradmin. The result may then be directly sent to /resources for actual creation.

formats: html , yaml , json

authentication:

oar

input: [resources] and [properties] entries are mandatory

structure: hash describing the resources to generate

fields:

- **resources** (*string*): A string corresponding to the resources definition as it could have been passed to the "oaradmin resources -a" command (see man oaradmin).
- **properties** (*hash*): an optional hash defining some common properties for these new resources

yaml example:

```
---
ressources: /nodes=node{2}.test.generate/cpu={2}/core={2}
properties:
  memnode: 1050
  cpufreq: 5
```

output: *structure*: an array of hashes containing the generated resources that may be pushed to POST /resources for actual creation

yaml example:

```
---
api_timestamp: 1321366378
items:
  - core: 1
    cpu: 1
    cpuset: 0
    network_address: node1.test.generate
  - core: 2
    cpu: 1
    cpuset: 1
    network_address: node1.test.generate
  - core: 3
    cpu: 2
    cpuset: 2
    network_address: node1.test.generate
  - core: 4
    cpu: 2
    cpuset: 3
    network_address: node1.test.generate
  - core: 5
    cpu: 3
    cpuset: 0
    network_address: node2.test.generate
  - core: 6
    cpu: 3
    cpuset: 1
```

```
      network_address: node2.test.generate
    - core: 7
      cpu: 4
      cpuset: 2
      network_address: node2.test.generate
    - core: 8
      cpu: 4
      cpuset: 3
      network_address: node2.test.generate
  links:
    - href: /oarapi-priv/resources/generate.yaml
      rel: self
  offset: 0
  total: 8
```

usage example:

```
# Generating new resources with curl
curl -i -X POST http://oar:kameleon@localhost/oarapi-priv/resources/gener
```

POST /resources

description: Creates a new resource or a set of new resources
formats: html , yaml , json
authentication:
oar
input: A [hostname] or [network_address] entry is mandatory

structure: A hash describing the resource to be created. An array of hashes may be given for creating a set of new resources. The result of a /resources/generate query may be directly posted to /resources.

fields:

- **hostname** alias **network_address** (*string*): the network address given to the resource
- **properties** (*hash*): an optional hash defining some properties for this new resource

yaml example:

```
---
hostname: test2
properties:
  besteffort: "NO"
  cpu: "10"
```

output: *structure*: hash returning the id of the newly created resource and status (or an array of hashes if a set of resources has been given on the input)

yaml example:

```
---
```

```
api_timestamp: 1245946199
id: 32
status: ok
uri: /resources/32
warnings: []
```

usage example:

```
# Adding a new resource with the ruby rest client (oar user only)
irb(main):078:0> r={ 'hostname'=>'test2', 'properties'=> { 'besteffort'=>
irb(main):078:0> puts post('/resources', r.to_json , :content_type => 'ap
```

POST /resources/<id>/state

description: Change the state
formats: html , yaml , json
authentication:
oar
input: A [state] entry is mandatory and must be "Absent", "Alive" or "Dead"

structure: hash of state

fields:

- **state**: Alive, Absent or Dead

yaml example:

```
---
state: Dead
```

output: *structure*:

yaml example:

```
---
api_timestamp: 1253283492
id: 34
status: Change state request registered
uri: /resources/34
```

usage example:

```
irb
```

DELETE /resources/<id>

description: Delete the resource identified by *id*
formats: html , yaml , json
authentication:
oar
output: *structure*: hash returning the status

yaml example:

```
---
api_timestamp: 1245946801
status: deleted
```

usage example:

```
# Deleting a resource with the ruby rest client
puts delete('/resources/32.yaml')
```

note: If the resource could not be deleted, returns a 403 and the reason into the message body.

DELETE /resources/<node>/<cpuset_id>

description: Delete the resource corresponding to *cpuset_id* on node *node*. It is useful when you don't know about the ids, but only the number of cpus on physical nodes.

formats: html , yaml , json

authentication: oar

output: *structure*: hash returning the status

yaml example:

```
---
api_timestamp: 1246459253
status: deleted
=> nil
```

usage example:

```
# Deleting a resource with the ruby rest client
puts delete('/resources/test/0.yaml')
```

note: If the resource could not be deleted, returns a 403 and the reason into the message body.

GET /admission_rules

description: Get the list of admission rules

formats: html , yaml , json

authentication: oar

output: *structure*: hash

fields:

- **items** : list of admission rules
- **links** : links to previous, current and next admission rules
- **offset** : current offset
- **total** : total of admission rules

yaml example:

```
---
items:
```

```
    - id: 1
      links:
         href: /admission_rules/1
         rel: self
      rule: 'if (not defined($queue_name)) {$queue_name="default";}'
    - id: 2
      links:
         href: /admission_rules/2
         rel: self
      rule: 'die ("[ADMISSION RULE] root and oar users are not allowed
    - id: 3
      links:
         href: /admission_rules/3
         rel: self
      rule: |2
            my $admin_group = "admin";
            if ($queue_name eq "admin") {
                  my $members;
                  (undef,undef,undef, $members) = getgrnam($admin_gr
                  my %h = map { $_ => 1 } split(/\s+/,$members);
                  if ( $h{$user} ne 1 ) {
                  {die("[ADMISSION RULE] Only member of the group "
                  }
            }
  links:
    - href: '/admission_rules.yaml?limit=5&offset=0'
      rel: previous
    - href: '/admission_rules.yaml?limit=5&offset=5'
      rel: self
    - href: '/admission_rules.yaml?limit=5&offset=10'
      rel: next
  offset: 5
  total: 5
```

usage example:

```
wget -q -O - http://localhost/oarapi/admission_rules.yaml

*note*: The following parameters can be passed through the requested URL
       - limit : limit of admission rules to be shown per page
       - offset : the page result offset
```

GET /admission_rules/<id>

description: Get details about the admission rule identified by *id*
formats: html , yaml , json

authentication:
oar
output: *structure*: 1 element array of hash

yaml example:

```
---
- id: 1
  links:
     href: /admission_rules/1
     rel: self
  rule: 'if (not defined($queue_name)) {$queue_name="default";}'
```

usage example:

```
wget -q -O - http://localhost/oarapi/admission_rules/1.yaml
```

DELETE /admission_rule/<id>

description: Delete the admission rule identified by *id*
formats: html , yaml , json
authentication:
oar
output: *structure*: hash returning the status

yaml example:

```
---
id: 32
api_timestamp: 1245946801
status: deleted
```

usage example:

```
# Deleting an admisssion rule with the ruby rest client
puts delete('/admission_rules/32.yaml')
```

note: **note:** Not all clients support the DELETE method, especially some www browsers. So, you can do the same thing with a POST of a {"method":"delete"} hash on the /admission_rule/<id> rule. If the admission rule could not be deleted, returns a 403 and the reason into the message body.

POST /admission_rules

description: Add a new admission rule
formats: html , yaml , json
authentication:
oar
input: *structure*: hash

fields:

- **rule** (*text*): The admission rule to add

yaml example:

```
---
rule: |
  echo "This is a test rule"
```

output: A 201 (created) header is returned if the rule is successfully created, with a location value.

yaml example:

```
---
api_timestamp: 1340180126
id: 19
rule: echo "This is a test rule"
uri: /oarapi-priv/admission_rules/19
```

POST /admission_rules/<id>

description: Update or delete the admission rule given by *id*
formats: html , yaml , json
authentication:
oar
input: *structure*: hash

fields:

- **rule** (*text*): The content of the admission rule to update
- **method=delete** : If given, the admission rule is deleted

yaml example:

```
---
rule: |
  echo "This is a test rule"
```

output: A 201 (created) header is returned if the rule is successfully updated, with a location value.

yaml example:

```
---
api_timestamp: 1340180126
id: 19
rule: echo"test rule"
uri: /oarapi-priv/admission_rules/19
```

GET /config

description: Get the list of configured variables
formats: html , yaml , json
authentication:
oar

output: *structure*: array of hashes

yaml example:

```
---
- id: DB_BASE_NAME
  links:
     href: /config/DB_BASE_NAME
     rel: self
  value: oar
- id: OARSUB_FORCE_JOB_KEY
  links:
     href: /config/OARSUB_FORCE_JOB_KEY
     rel: self
  value: no
- id: SCHEDULER_GANTT_HOLE_MINIMUM_TIME
  links:
     href: /config/SCHEDULER_GANTT_HOLE_MINIMUM_TIME
     rel: self
  value: 300
- id: SCHEDULER_RESOURCE_ORDER
  links:
     href: /config/SCHEDULER_RESOURCE_ORDER
     rel: self
  value: 'scheduler_priority ASC, suspended_jobs ASC, network_address
- id: SCHEDULER_PRIORITY_HIERARCHY_ORDER
  links:
     href: /config/SCHEDULER_PRIORITY_HIERARCHY_ORDER
     rel: self
  value: network_address/resource_id
- id: OARSUB_NODES_RESOURCES
  links:
     href: /config/OARSUB_NODES_RESOURCES
     rel: self
  value: network_address
- id: SCHEDULER_JOB_SECURITY_TIME
  links:
     href: /config/SCHEDULER_JOB_SECURITY_TIME
     rel: self
     value: 60
- id: DETACH_JOB_FROM_SERVER
  links:
     href: /config/DETACH_JOB_FROM_SERVER
     rel: self
  value: 0
```

```
- id: LOG_LEVEL
  links:
     href: /config/LOG_LEVEL
     rel: self
  value: 2
- id: OAREXEC_DEBUG_MODE
  links:
     href: /config/OAREXEC_DEBUG_MODE
     rel: self
  value: 0

   .....
   .....
```

usage example:

```
curl -i -X GET http://login:password@localhost/oarapi-priv/config.yaml
```

GET /config/file

description: Get the raw config file of OAR. It also output the path of the file used by the API.

formats: html , yaml , json

authentication: oar

output: *structure*: hash

fields:

- **path** : The path of the config file
- **file** : The raw content of the config file (text)

usage example:

```
curl -i -X GET http://kameleon:kameleon@localhost/oarapi-priv/config/file
```

GET /config/<variable>

description: Get details about the configuration variable identified by *variable*

formats: html , yaml , json

authentication: oar

output: *structure*: 1 element array of hash

yaml example:

```
---
- id: DB_TYPE
  links:
     href: /config/DB_TYPE
     rel: self
  value: mysql
```

usage example:

```
curl -i -X GET http://login:password@localhost/oarapi-priv/config/DB_TYPE
```

POST /config/<variable>

description: Change the value of the configuration variable identified by *variable*
formats: html , yaml , json
authentication:
oar
input: A [value] entry is mandatory

structure: hash describing the new value of the variable

fields:

- **value** (*string*): the value of the given variable

yaml example:

```
---
value: 'state=Finishing,Running,Resuming,Suspended,Launching,toLaunch
```

output: *structure*: hash returning the variable and his new value

yaml example:

```
---
    API_JOBS_URI_DEFAULT_PARAMS:
      value: 'state=Finishing,Running,Resuming,Suspended,Launching,to
```

usage example:

```
curl -i -X POST http://login:password@localhost/oarapi-priv/config/API_JO
```

note: config.yaml contains the value of the variable.

GET /media/ls/<file_path>

description: Get a list of the directory from the path given by *file_path*. The *file_path* may contain the special character "~" that is expanded to the home directory of the user that is making the request.
formats: html , yaml , json
authentication:
user
output: *structure*: array of hashes giving for each listed file: the name, the mode, the size, the modification time and the type (*f* for a file or *d* for a directory)

yaml example:

```
---
api_timestamp: 1340095354
items:
  - mode: 33188
    mtime: 1339685040
    name: API.pm
```

```
    size: 58620
    type: f
  - mode: 16877
    mtime: 1340094660
    name: bart
    size: ~
    type: d
  - mode: 16877
    mtime: 1338993000
    name: cigri-3
    size: ~
    type: d
  - mode: 16877
    mtime: 1340095200
    name: oar
    size: ~
    type: d
  - mode: 16877
    mtime: 1334132940
    name: oar_install
    size: ~
    type: d
  - mode: 33261
    mtime: 1339685040
    name: oarapi.pl
    size: 75939
    type: f
  - mode: 33261
    mtime: 1340027400
    name: test.sh
    size: 43
    type: f
links:
  - href: /oarapi-priv/media/ls/~/
    rel: self
offset: 0
total: 7
```

usage example:

```
curl -i -X GET http://kameleon:kameleon@localhost/oarapi-priv/media/ls/~/
```

note: returns a 404 if the path does not exist, or a 403 if the path is not readable.
Errors in debug mode (with ?debug=1) are formated into yaml.

GET /media/<file_path>

description: Get a file located on the API host, into the path given by *file_path*. The *file_path* may contain the special character "~" that is expanded to the home directory of the user that is making the request.

parameters:
- **tail**: specifies an optional number of lines for printing only the tail of a text file

formats: application/octet-stream

authentication: user

output: octet-stream

usage example:

```
curl -i -H'Content-Type: application/octet-stream'  http://kameleon:kamel
```

note: returns a 404 if the file does not exist, or a 403 if the file is not readable. Errors in debug mode (with ?debug=1) are formated into yaml.

POST /media/<file_path>

description: Upload or create a file on the API host, into the path given by *file_path*. The *file_path* may contain the special character "~" that is expanded to the home directory of the user that is making the request. If the path does not exists, the directories are automatically created. If no data is passed, an empty file is created. If binary data is sent as POSTDATA, then it is a file to upload.

formats: application/octet-stream

authentication: user

output: 201 if ok

usage example:

```
curl -i -X POST -H'Content-Type: application/octet-stream' --data-binary
```

POST /media/chmod/<file_path>

description: Changes the permissions on a file: do a chmod(1) on *file_path*. The special character "~" is expanded as the home of the user that makes the query.

formats: html , yaml , json

authentication: user

input: A [mode] entry is mandatory

mode: A mode definition as passed to the "chmod" unix command.

output: 202 if ok

usage example:

```
curl -i -X POST http://kameleon:kameleon@localhost/oarapi-priv/media/chmo
```

DELETE /media/<file_path>

description: Delete the file or directory given by *file_path*. The *file_path* may contain the special character "~" that is expanded to the home directory of the user that is making the request. If the path is a directory, then it is deleted recursively.
formats: application/octet-stream
authentication:
user
output: 204 if ok
usage example:

```
curl -i -X DELETE -H'Content-Type: application/octet-stream' http://kamel
```

Some equivalences with oar command line

OAR command	**REST request**
oarstat	GET /jobs.html
oarstat -Y	GET /jobs/details.yaml?structure=oar
oarstat -Y -fj <id>	GET /jobs/<id>.yaml
oardel <id>	DELETE /jobs/<id>.yaml
oardel <id> *(alternative way)*	POST /jobs/deletions/<id>/new.yaml
oarnodes -Y	GET /resources/full.yaml?structure=oar
oarnodes -Y -r1	GET /resources/1.yaml?structure=oar