

OASYS v3.3.15

API reference

Author: Eric J. FRANCOIS

June 5, 2024

1 general information

OASYS provides a number of APIs that allow external sources to fetch data from OASYS, trigger OASYS to show the preview of an item or even to inject information (e.g. creation of test takers controlled from an outside source).

While some APIs, like previewing an item, require to work with a permanent link that can be sent via e-mail for review, the APIs that actually modify the data inside OASYS (e.g. creation of back end users) require a higher security, to make sure a url including its parameters can only be executed once and only one specific instance of OASYS.

For this reason there are two different types of APIs: the high security and the low security ones. Both APIs have one thing in common: Before they can be used, an API key needs to be created in the database table **apiKeys**, for the application that wants to use OASYS. This ensures that only known sources can use a specific instance of OASYS. The api key is linked to one specific API, so that an application does not automatically get permission to use every available API. Each permission needs to be given by a new entry in the **apiKeys** table.

The API URL is always:

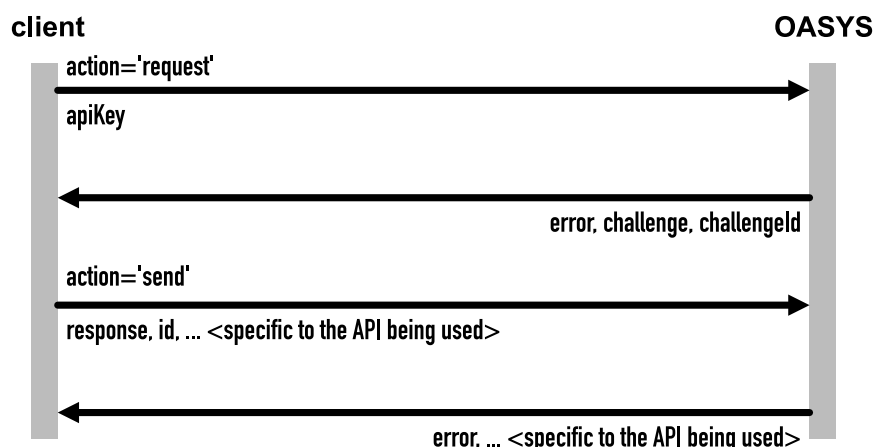
`http(s)://<baseurl of OASYS>/api/<name of API to use>`

For example:

`https://demo.oasys.lu/api/createUser/`

2 high security APIs

In addition to the API key, a high security API requires a secret to be specified in the **apiKeys** table. This secret needs to be known to the application using the API. The secret will never be transmitted.



As seen in the chart, there are a few steps to use this kind of API:

1. The client needs to request the use of the API by transmitting its API key. The following GET or POST parameters need to be sent:
 - **action** = 'request'
 - **key** = <the API key>

2. The API now saves the request in the **apiRequests** table with the current timestamp. The request will only be valid for the next minute. OASYS sends back a JSON object that will contain a key called **error** with a string if something went wrong (e.g. unknown API key), or if everything succeeded, it will contain the two keys **challenge** and **challengeld**.

The **challenge** key will contain a string necessary to create a valid response and the **challengeld** key contains an integer which identifies the request in the **apiRequests** table.

3. The client will now calculate a proper response by concatenating the secret and the challenge and then hashing the string with an algorithm supported by PHP (by default BCRYPT since PHP 5.5). Then it will contact the API again—within 1 minute—and transmit the following GET or POST parameters:

- **action** = 'send'
- **response** = <the hashed response>
- **id** = <the request id that was transmitted by the client>
- ...any other information necessary for the particular API being used

4. The API will now check if the request still exists and if the response to the challenge is valid. Then it will delete the request from the table, so that it cannot be used again later with the exact same parameters.

If everything checks out, the rest of the parameters are being checked by the API then it returns another JSON object that may contain an **error** or if everything succeeded the expected data (check specific APIs for the exact data to expect).

3 low security APIs

Currently there are no low security APIs yet in OASYS

4 specific API details

4.1 createUser

API type: high security

This API allows to create a backend user (e.g. to give demo access to the editor of OASYS). The client needs to send a login for the user and the name of an existing usergroup which the user should be added to. Optionally a password can be specified for the new user. If omitted, a password will be generated randomly and sent back.

input parameters:

username *	<i>string</i>	the login to be used—this needs to be no longer than 32 characters and may contain letters, numbers, understores, hyphens and periods.
usergroup *	<i>string</i>	the name of the group to add the user to—this needs to exist already, otherwise the creation will fail.
password	<i>string</i>	the password to be used—this must not exceed 32 characters

*mandatory parameters

returned data:

error	error message if something went wrong
username	the login that was requested
usergroup	the usergroup that was sent to the API
password	the password that was requested, respectively the randomly generated password

4.2 createTestTaker

API type: high security

This API allows the creation of a list of test takers (with login, meta data, password, password tag, one or more tests to be linked) in an existing folder. Everything is optional except for the login and the test(s) to be linked. All parameters other than the logins can either be passed globally to be applied to all test takers (e.g. a common password for the whole class) or individually for each test taker. If both an individual parameter and a global one is set, the individual one is being used.

input parameters:

testTakers *	<i>JSON</i>	array of test takers to create, see below for details on this array
password	<i>string</i>	a global password to be used for all test takers
testIds	<i>JSON</i>	array of test ids to be linked with every login—these are the internal ids of tests and they must exist in order to be linked
folderId	<i>integer</i>	the id of the folder all the test takers are to be created in
metaData	<i>JSON</i>	array of key-value pairs of meta data to be added globally to every test taker (e.g. name of the school)
tag	<i>string</i>	a tag to be applied to all passwords created

*mandatory parameters

keys in each test taker in the array:

login *	<i>string</i>	the actual login to be entered
password	<i>string</i>	a password to be used for this test taker—if omitted, a random password will be created and returned
testIds *	<i>array</i>	array of test ids to be linked with this login—these are the internal ids of tests and they must exist in order to be linked
folderId	<i>integer</i>	the id of the folder the test taker is to be created in
metaData	<i>array</i>	array of key-value pairs to be added to the test taker
tag	<i>string</i>	a tag to be applied to the current password

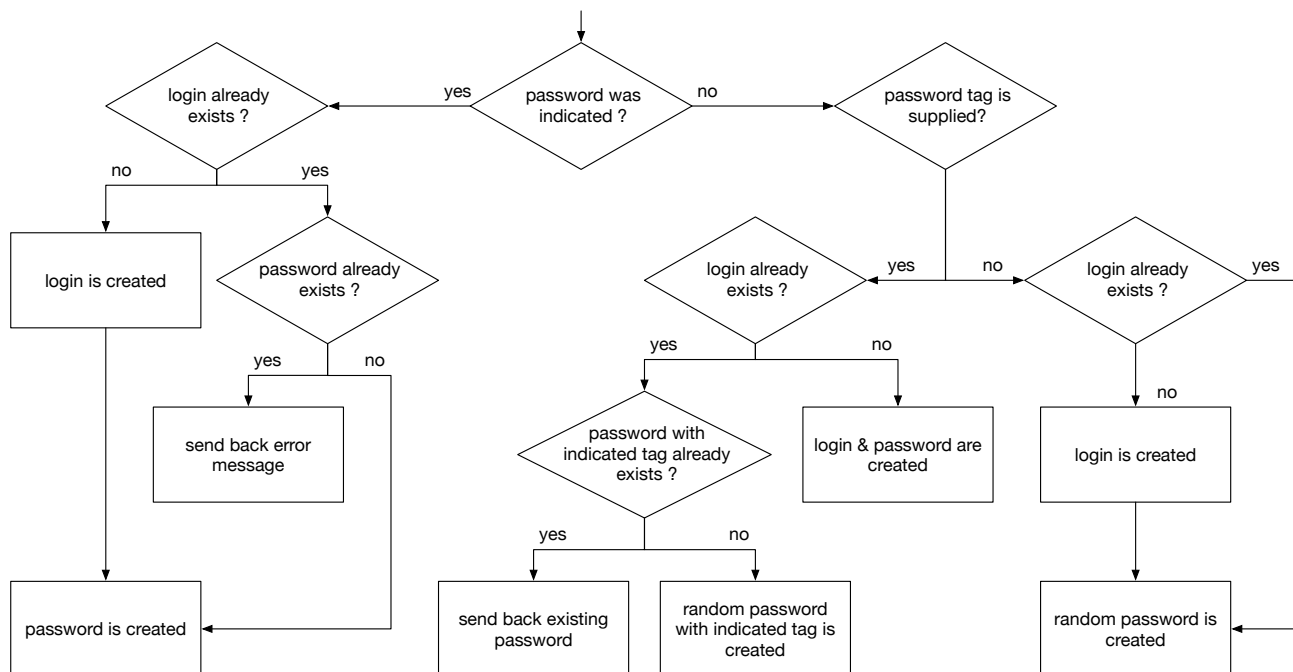
*mandatory parameters

Attention: While the arrays **testIds** and **metaData** need to be JSON encoded when sent as a global parameter, they must not be JSON encoded when part of a test taker as the **testTakers** array already is JSON encoded—there is no use encoding it twice.

returned data:

error	error message if something went wrong
testTakers	the same array that was sent to the API with an additional error field with a message if anything did not go according to plan. If no password was specified it will also contain the password (either a randomly generated one, or a previously existing one from the database) There is also an extra field called state which will be true if the current credentials are available for login and false if the credentials already existed and have been used, so that no further login is possible any longer.

Depending on which parameters are sent, this API adopts a different behaviour:



4.3 getProgress

API type: high security

This API allows checking the progress of a specific login and test combination. If the test taker has filled in the same test multiple times (by the means of different passwords), the latest password will be taken (the password with the highest auto increment id). If more passwords exist, linking the test taker to the exact same test, it is possible to distinguish the passwords by the use of tags, which can optionally be indicated here as well.

input parameters:

login *	<i>string</i>	the login for which the progress is requested
testId *	<i>integer</i>	the internal id of the test for which the progress is requested
tag	<i>string</i>	the tag of the password we are looking for—if the tag is null or not sent at all, it will be ignored.

*mandatory parameters

returned data:

error	error message if the request failed—false if everything went smoothly
status → started	0 if test taker never logged in to this test; 1 if there has been activity
status → finished	0 if the test taker can still log in and change data; 1 if the test has been finalised (e.g. answers were submitted respectively the time ran out)
total	the total count of fields in this test
answers	the count of fields to which the test taker answered
percentage	percentage of fields that were filled

Attention: Beware of tests with conditional branching. There is no way to predict exactly which pages were hidden from the test taker at this point, so the total number of fields may be higher than those that were really available to the test taker.

4.4 getTestProgress

API type: high security

This API allows checking the progress of a specific test for all logins that have been active in this test. If a test taker has filled in the same test multiple times (by the means of different passwords), the latest activity will be taken (based on the last login timestamp). Furthermore it is possible to define two types of filters: you can define to only get the progress of passwords with a specific tag and you can define a cutoff date in order to ignore any data collected before that date.

input parameters:

testId *	<i>integer</i>	the internal id of the test for which the progress is requested
filters	<i>JSON</i>	an array which can define additional filters—the keys 'tags' and 'cutoffDate' are supported here
filters → tags	<i>array</i>	list of strings that are considered acceptable password tags
filters → cutoffDate	<i>string</i>	a date in the form YYYY-MM-DD in order to ignore any data gathered before this date.

*mandatory parameters

returned data:

error	error message if the request failed—false if everything went smoothly
activity	array of logins and their progress data, see below for details
total	the total count of fields the test contains

returned activity data (in activity → <login>):

login	login string (duplicate of the key)
tag	password tag, if any
started	0 if the test taker never logged in to this test, 1 if there has been activity
finished	0 if the test taker can still log in and change data; 1 if the test has been finalised (e.g. answers were submitted respectively the time ran out)
answers	the count of fields to which the test taker answered
lastAnswer	timestamp of the last answer given in this test
percentage	percentage of fields that were filled

Attention: Beware of tests with conditional branching. There is no way to predict exactly which pages were hidden from the test taker at this point, so the total number of fields may be higher than those that were really available to the test taker.

4.5 editTestTaker

API type: high security

This API allows making modifications to the activity of a test taker, resp. to the test taker itself. It can change the time left for a test, reset the activity entirely, set current page of a test taker or rename the login of a test taker.

input parameters:

login *	string	the login to be edited
testId *	integer	the internal id of the test for which activity is to be edited
tag	string	the tag of the password we are looking for—if the tag is null or not sent at all, it will be ignored.
tasks *	JSON	array of tasks to be executed, whereas each task consists on an array with the property task describing the action to be done and if necessary a property value (see below for details)

*mandatory parameters

defined keywords for the 'task' property:

resetActivity	deletes all activity for login and testId combination, no value required
setTimeLeft	change the time left, value property must be an integer (time in seconds, or -1 for no limit)
setCurrentPage	set the page to be shown to the test taker after next login, value property must be an integer (start counting at 0)
changeLogin	rename the login, value property must be a string with the new name

returned data:

error	error message if the request failed—false if everything went smoothly
-------	---

notes:

- Even though renaming a login does not require a testId, it is still required as a sanity check. If a login is indicated with a testId that does not exist, we have to assume that the request is a mistake (or worse, an attack).
- When using `setTimeLeft` with value -1 this cannot override a time limit set in the test. In this case the test taker will get the maximum allowed time.
- If `setTimeLeft` is used with a value higher than 0, it is possible to give more time than the test allows. This is useful if a test taker has to be given extra time due to a handicap.
- Renaming a login will not allow to rename it to an already existing login. If a login is specified that exists, the request will fail with an error message.
- The `setCurrentPage` task should be used in combination with `setTimeLeft` when reopening an already submitted test. If the time left was 0 and is now set to -1 (or any positive value), the test taker would normally see the last page with the submit button when logging in again. Here it is advisable to set the current page to 0, so that the test taker will see the first page again by default.

5 PHP example of using a high security API

```
<?php
$protocol = "https://";
$url = "demo.oasys.lu/oasys/api/getTestProgress/";

$apiKey = 'getTestProgressDemo';
$secret = 'demo';
$testId = 42;
$filters['tags'] = ['demotag'];
$filters['cutoffDate'] = '2023-05-01';

$data = file_get_contents($protocol . $url . "?action=request&key=$apiKey");
$data = json_decode($data);

if ($data->error) {
    print_r($data);
    die();
}

$response = password_hash($secret . $data->challenge, PASSWORD_DEFAULT);
$id = $data->challengeId;

$postData = http_build_query([
    'action' => 'send',
    'response' => $response,
    'id' => $id,
    'testId' => $testId,
    'filters' => json_encode($filters)
]);

$opts = [
    'http' => [
        'method' => 'POST',
        'content' => $postData,
        'ignore_errors' => true,
        'header' => 'Content-type: application/x-www-form-urlencoded',
        'ssl' => [
            'verify_peer' => false,
            'verify_peer_name' => false
        ]
    ]
];

$context = stream_context_create($opts);
$raw = file_get_contents($protocol.$url, false, $context);

$data = json_decode($raw, true);

if (json_last_error() != JSON_ERROR_NONE) {
    echo "<p>Error decoding answer (JSON error " . json_last_error() . ")!</p>";
    echo "<pre>$raw</pre>";
    die();
}

echo "<pre>";
print_r($data);
echo "</pre>";
```