Recruiting Technology

Let's seize opportunities.

Together

Manual OWS webhooks

Version 1.3

A manual for external parties integrating with OTYS to configure webhooks which will be triggered after specific OWS actions are executed, to further improve their (current) integration.

Date: 03-10-2022

Version management

Below you will find an overview of the versions available for this document.

Version	Date	Author	Description
1.0	04-12-2020	Bastiaan Brans	Created initial version of document, including four chapters: 'General' (chapter 1), 'Activation & configuration' (chapter 2), 'Process webhook requests' (chapter 3) & 'Logging' (chapter 4).
1.1	15-12-2020	Bastiaan Brans	Description of additional fields 'klantid', 'userid' and 'klantGlobal' in the response of the getList method in paragraph 2.3.1. Description of the additional field 'klantGlobal' in the request of the add & update methods in paragraph 2.3.2 & 2.3.3. Changed the way we output the users request & response in paragraphs 3.1 & 3.2. Added an example on how to easily prevent loops in paragraph 3.3. Added additional (optional) parameter to define number of desired results in paragraph 4.3.
1.2	15-03-2021	Bastiaan Brans	Since logging is globally enabled, removed the comment about requesting to enable logging in paragraph 4.1
1.3	03-10-2022	Bastiaan Brans	We have added a new field 'dt_processed' to the response of 'Otys.Services.WebhooksService.getLogs'. Updated 'Logged data' (paragraph 4.2) and 'Accessing logs' (paragraph 4.3) accordingly.

Contents

1. General

In this chapter we will provide you some generic information about OTYS, OTYS web services (OWS), the OWS webhooks and what you can expect from this document.

1.1 OTYS Recruiting Technology

OTYS Recruiting Technology (hereafter 'OTYS') develops, maintains and optimizes software for around 1.000 clients worldwide. The core of this software is created for recruiting purposes (corporate recruitment, agencies & job boards), but it has various modules for the 'steps after recruitment' (assignment registration, timesheet registration & invoicing) as well.

1.2 OTYS Web Services (OWS)

OTYS has very extensive web services to connect to the OTYS system. Where other similar parties have web services as an 'add on' to their own application (with normally limited capabilities), OTYS chose to create a web service layer which is an integral part of their current software solution. All actions users take through the user interface (OTYS Go!) are done by OWS request, thus allowing external parties to connect with the web services and (technically) be able to do all the things end users are able to do through the GUI.

1.3 OWS webhooks

If an external party connects to OTYS using OWS, there is always some kind of 'trigger' to exchange data. This trigger can be (from the OTYS perspective) an 'external trigger' (for example a visitor of an external website opens a vacancy detail page, which triggers the external party to retrieve the vacancy information using OWS), but it can also be an 'OTYS trigger' (for example a candidate main status in OTYS has been changed, which should trigger the external party to send an assessment). In case of an OTYS trigger, the only way external parties were able to know 'when to come into action' is to set up an automated process which runs periodically (for example a CRON which runs every hour) which searches for specific records in OWS (for example list all candidates with status 'Send assessment') and take action upon it (for example send the assessment and update the candidate status in OTYS to 'Assessment sent'). Since this is not an ideal situation (it causes a delay and extra traffic), we created 'OWS webhooks'.

A webhook is an 'active signal' to another application, that some event has occured. This way we are able to provide external parties an active signal that for example a candidate has been updated, so that the external party (being you) can take follow up actions.

It is possible to configure webhooks based upon data service / method combinations and for each webhook a specific URL can be configured. This way a you can for example configure that if CandidateService.update is triggered (thus a candidate is updated in OTYS), the webhook URL https://www.externalparty.com/otys/webhooks/candidate/update is triggered. This trigger will 'post' the OWS request and response, to give you as an external party more details about the action in the OTYS system. Based upon this trigger you can take the appropriate follow up action.

1.4 About this document

This document describes the concept of OWS webhooks and how external parties can use them in their processes. This document will not describe the generic working of OWS and is not a document where you learn how to develop these kinds of integrations in general. It is therefore intended for external parties already connecting to OTYS using OWS and want to improve this integration by implementing webhooks.

Although we do our best to help your brain convert this information into the usage of the OWS webhooks for the topics you want to implement it (for example by providing examples), your brain will still need to do the hard work by figuring out how your process can use these OWS webhooks in an optimal way.

2. Activation & configuration

In this chapter we will explain how the webhooks can be activated and configured. Please note that after you have completed this chapter you will be able to add webhooks which trigger endpoints on your end. In the next chapter we will explain how you are able to process the requests send to your endpoint.

2.1 Activation

Webhooks are currently enabled by default. To start using the webhooks you can simply start adding one or multiple webhooks as explained below.

IMPORTANT: Please note that to avoid performance issues, webhooks are cached. Creating new webhooks, updating existing webhooks and deleting existing webhooks can take up to 10 minutes to take effect.

2.2 Define service calls

To make a generic solution for webhooks, we have made them dependent on OWS service calls. OWS has a huge list of 'data services' (for example 'CandidateService' for basic candidate operations) and each data service consists of various 'methods' (for example 'update' to update a record). Based upon the combination of these (for example ‘CandidateService.update’, called ‘service calls’) OWS requests are made. The downside of this is that you will need to pinpoint the service call (in a case of a candidate update this can be ‘CandidateService.update, but it can also be something different). We will help you pinpoint the correct service call in a bit. The upside of this generic approach is however that the solution is very flexible (which will help you in the long run).

As mentioned before, the OTYS Go! application is based upon OWS. This means OTYS Go! sends (JSON) web service requests for all actions taken in the system. Using a web debugging proxy tool such as Fiddler (https://www.telerik.com/fiddler) you are able to intercept & inspect these requests.

Before you start configuring your webhooks, you will need to define which processes in OTYS Go! should trigger these webhooks. This will most probably start by asking your client what should trigger the process (or define it for yourself) and then execute the action in OTYS Go! (while Fiddler or a different solution is intercepting your requests). You will then see the actual OWS requests that are being executed and are able to pinpoint the correct service call (for example CandidateService.update).

Please note a field in OTYS Go! can many times be changed in multiple ways. A candidate main status can for example be changed from a candidate detail, but also in batch from the candidate list, from workflows and from other places. Make sure to go through the possible ‘user actions’ with your client that should trigger the webhook, since they might use alternative service calls.

2.3 Configuration

Now you know which service call should trigger your webhook, it is time to configure them. We currently do not have a user interface yet for managing these webhooks, but since you are IT nerds (like us) and know how to communicate using OWS; we expect you to be happy (for now) being able to manage them using OWS itself.

IMPORTANT: Please note that to avoid performance issues, webhooks are cached. Creating new webhooks, updating existing webhooks and deleting existing webhooks can take up to 10 minutes to take effect.

2.3.1 List existing webhooks

You can retrieve a list of the currently configured webhooks by sending the following request to OWS, where **[sessionid]** is the session ID your session you received by providing your API key:

```
{
  "jsonrpc": "2.0",
  "method": "Otys.Services.WebhooksService.getList",
  "params": [
    "[sessionid]"
  ],
  "id": 1
}
```

If a valid session ID has been provided, this will result in the following response:

```
{
    "result": [
        {
            "uid": [uid],
            "serviceCall": "[servicecall]",
            "webhookUrl": "[webhookurl]",
            "klantid": "[klantid]",
            "userid": "[userid]",
            "klantGlobal": "[klantglobal]"
        }
    ],
    "id": 1
}
```

Please note the result object consists a recursive array. So if there are five webhooks configured you will see five arrays.

The response consists of the following fields per webhook:

- **[uid]**: The UID of the webhook.
- **[servicecall]**: The OWS service call that should trigger the webhook (combination between the data service & method), for example 'CandidateService.add'.
- **[webhookurl]**: The webhook URL that is being called if the OWS service call is executed.
- **[klantid]**: The OTYS client ID connected to the webhook.
- **[userid]**: The OTYS user ID connected to the webhook.
- **[klantglobal]:** If this is set to 'true', the webhook can be triggered by all users of the client (normally the desired behavior). If this is set to 'false' the webhook can only be triggered by the user who created the webhook (which is mainly used for testing purposes) as defined in

[userid]. If the same service call has multiple webhooks, we will only execute one and we will prefer the 'user specific' one; making the testing process easier.

2.3.2 Create new webhooks

When creating a new webhook, you should include the following fields:

- **[sessionid]**: The session ID of your session you received by providing your API key.
- **[servicecall]**: The OWS service call that should trigger the webhook (combination between the data service & method), for example 'CandidateService.add'.
- **[webhookurl]**: The webhook URL that is being called if the OWS service call is executed. In this URL you can use the following merge fields which will be merged by OTYS:
  - **{%user.clientId%}**: This will merge into the client ID of the user which executed the OWS request that triggered the webhook.
  - **{%user.internalId%}**: This will merge into the user ID of the user which executed the OWS request that triggered the webhook.
- **[klantglobal]:** If this is set to 'true', the webhook can be triggered by all users of the client (normally the desired behavior). If this is set to 'false' the webhook can only be triggered by the user who created the webhook (which is mainly used for testing purposes) as defined in

To 'know' which service call to use, you will probably take the desired (trigger) actions in OTYS Go! and use an HTTP sniffer like Fiddler to intercept your requests and pinpoint the right service call (see paragraph 2.2).

You can then create a new webhook by sending the following request to OWS:

```
{
  "jsonrpc": "2.0",
  "method": "Otys.Services.WebhooksService.add",
  "params": [
    "[sessionid]",
    {
      "serviceCall": "[servicecall]",
      "webhookUrl": "[webhookurl]",
      "klantGlobal": "[klantglobal]"
    }
  ],
  "id": 1
}
```

If a valid session ID has been provided, this will result in the following response:

```
{
  "result": "[uid]",
  "id": 1
}
```

This **[uid]** is the UID of the webhook that has been created.

2.3.3 Update existing webhooks

When updating an existing webhook, you should include the following fields:

- **[sessionid]**: The session ID of your session you received by providing your API key.
- **[uid]**: The UID of the webhook you would like to update.
- **[servicecall]**: The OWS service call that should trigger the webhook (combination between the data service & method), for example 'CandidateService.add'.
- **[webhookurl]**: The webhook URL that is being called if the OWS service call is executed. In this URL you can use the following merge fields which will be merged by OTYS:
    - {%user.clientId%}: This will merge into the client ID of the user which executed the OWS request that triggered the webhook.
    - {%user.internalId%}: This will merge into the user ID of the user which executed the OWS request that triggered the webhook.
- **[klantglobal]:** If this is set to 'true', the webhook can be triggered by all users of the client (normally the desired behavior). If this is set to 'false' the webhook can only be triggered by the user who created the webhook (which is mainly used for testing purposes) as defined in

You can then update the existing webhook by sending the following request to OWS:

```
{
  "jsonrpc": "2.0",
  "method": "Otys.Services.WebhooksService.update",
  "params": [
    "[sessionid]",
    [uid],
    {
      "serviceCall": "[servicecall]",
      "webhookUrl": "[webhookurl]",
      "klantGlobal": "[klantglobal]"
    }
  ],
  "id": 1
}
```

If a valid session ID & webhook UID has been provided, this will result in the following response:

```
{
  "result": [],
  "id": 1
}
```

The webhook has now been updated correctly.

2.3.4 Delete existing webhooks

When deleting an existing webhook, you should include the following fields:

- **[sessionid]**: The session ID of your session you received by providing your API key.
- **[uid]**: The UID of the webhook you would like to delete.

You can then delete an existing webhook by sending the following request to OWS:

```
{
  "jsonrpc": "2.0",
  "method": "Otys.Services.WebhooksService.delete",
  "params": [
    "[sessionid]",
    [uid]
  ],
  "id": 1
}
```

If a valid session ID & webhook UID has been provided, this will result in the following response:

```
{
  "result": true,
  "id": 1
}
```

The webhook has now been deleted correctly.

3. Process webhook requests

In chapter 2 you found out how you can create & manage your webhooks, so we expect you to have configured at least one webhook which sends a request to an endpoint of your own. In this chapter we will explain how you can process these triggered webhooks.

3.1 General structure

If a webhook is triggered, we send both the original OWS request that triggered the webhook as our response. Please note that due to security reasons we will filter out some information (like login requests and session IDs).

Since the webhook will be triggered on some generic event (for example an updated candidate) and your process will probably need more details (only trigger follow up action if status is set to a specific status and then execute actions for this specific candidate) your script will need to retrieve this data and process it in a logical way that serves the purpose you want to achieve.

We will give you some examples below to make this a bit more 'liveley'.

3.2 Examples

Below you will find some examples in PHP to process your result. Of course if you are using a different programming language it will work a bit different for you (but we expect that these examples will help you anyway).

3.2.1 Echo posted data

Let's say you have configured a webhook which should be triggered based upon candidate data being changed somewhere in OTYS Go! You have found out the service call for this is CandidateService.update and configured a webhook (see chapter 2) with the webhook URL https://www.yourdomain.com/otys/webhooks/candidateUpdated.php.

To simply echo the data send by OTYS you can create a very simple PHP file which looks as follows:

```
<?php

// Retrieve JSON
$dataJson = file_get_contents('php://input');

// Echo JSON
echo $dataJson;

?>
```

Please note this will echo the data to the body. Since the script will be executed by our server, you will need to have some way of accessing it (whatever works best for you).

3.2.1 Echo URL parameters

As mentioned before, it is possible to include merge fields in the webhook URL. Additionally you can choose to let one PHP file process all your webhooks and by using (webhook specific) parameters. So let's say that instead of the webhook URL of the previous example, you have used the following URL: https://www.yourdomain.com/otys/webhook.php?action=candidateUpdated&client={%user.clientId%}&user={%user.internalId%}

To simply retrieve the URL parameters you can create a very simple PHP file which looks as follows:

```
<?php

// Define variables
$action = $_GET['action'];
$client = $_GET['client'];
$user = $_GET['user'];

// Echo action
if($action == 'candidateUpdates') {
      echo "Candidate was updated by ";
      echo "client " . $client;
      echo "and user " . $user;
      }

?>
```

Please note this will also echo the parameters to the body. Since the script will be executed by our server, you will need to have some way of accessing it (whatever works best for you).

3.2.3 Execute action if vacancy is published

Let's say you have configured a webhook which should be triggered based upon vacancy detail being changed somewhere in OTYS Go! You have found out the service call for this is VacancyService.update and configured a webhook (see chapter 2) with the webhook URL https://www.yourdomain.com/otys/webhooks/publishVacancy.php which should update the vacancy in the external database of your non OTYS website.

Your PHP code could look something like this:

```
<?php

// Retrieve JSON
$dataJson = file_get_contents('php://input');

// Decode JSON into array
$dataArray = json_decode($dataJson, true);

// Set vacancy UID & vacancy published variables
$vacancyUid = $dataArray['request']['args'][1];
$vacancyPublished = $dataArray['request']['args'][2]['published'];

// Execute action if vacancy is published
if ($vacancyPublished == true) {

      // Place code here to publish vacancy in external database

} else {

      // Place code here to unpublish vacancy in external database

}

?>
```

As you can probably see, the script is:

1. Retrieving the JSON data from the body;
2. Decoding the JSON into a PHP array;
3. Retrieving the vacancy UID and the vacancy publication from the array and placing them in variables;
4. Checking if the vacancy is set to published or unpublished and execute follow up actions (update the external database) if this is the case.

As mentioned before this is just an example of how you can implement something like this. Feel free to choose a different approach which you feel is best.

3.2.4 Execute action if candidate status is set to ‘Send assignment’

Let’s say you have configured a webhook which should be triggered based upon candidate detail being changed somewhere in OTYS Go! You have found out the service call for this is CandidateService.updateEx and configured a webhook (see chapter 2) with the webhook URL https://www.yourdomain.com/otys/webhooks/sendAssessment.php which should send an assessment to the candidate if the candidate status is set to ‘Send assessment’ and update the status to ‘Awaiting assessment’.

Your PHP code could look something like this:

```
<?php

// Retrieve JSON
$dataJson = file_get_contents('php://input');

// Decode JSON into array
$dataArray = json_decode($dataJson, true);

// Set candidate UID & candidate status ID variables
$candidateUid = $dataArray['request']['args'][1];
$candidateStatusId = $dataArray['request']['args'][2]['statusId'];

// Execute action if candidate status is 'Send assessment' (status ID
12345)
if ($candidateStatusId == 12345) {

      // Place code here to send assessment

      // Place code here to update candidate status to 'Awaiting
assessment' using OWS

}

?>
```

As you can probably see, the script is:

1. Retrieving the JSON data from the body;
2. Decoding the JSON into a PHP array;
3. Retrieving the candidate UID and the candidate status from the array and placing them in variables;
4. Checking if the candidate status ID is ‘12345’ (which will of course be a different status ID in your case) and execute follow up actions (send the assessment & update the status) if this is the case.

As mentioned before this is just an example of how you can implement something like this. Feel free to choose a different approach which you feel is best.

3.3 Preventing loops

Please note that webhooks are triggered by OWS requests. Because there is a good chance your script will do additional OWS request, you should ensure your script to create loops.

If for example your webhook is triggered by service call CanddiateService.update and the script which is triggered does an OWS request using service call CanddiateService.update, it will trigger the script again (and again and again and..).

Please think through your end-to-end solution and prevent these loops occurring. An easy way of doing so is to kill your script if the the OTYS user connected to your API key (thus 'you') is doing the request that triggered the webhook:

```
<?php

// Define variables
$user = $_GET['user'];

// Die if user triggering script is me
if($user == 12345) {
      die("message");
}

// Execute rest of script

?>
```

Of course this is just a suggestion. Based upon your specific scenario you might want to implement a different method of preventing loops.

3.4 Limitations

To avoid possible issues caused by these loops we have maximized the number of webhooks to be triggered by 1.000 per day. If this is causing any issues for your intended integration, please contact us so we can find an appropriate solution.

4. Logging

To be able to debug the webhook process, we have included a logging functionality where we can log triggered webhooks (and your scripts response) on our end. These logs have also been made available to you using OWS.

4.1 Activation

Logging of triggered webhooks is enabled for all webhooks.

4.2 Logged data

When logging is enabled, we will be logging the following data:

- Log ID
- Webhook URL
- OWS request that triggered the webhook
- OWS response that triggered the webhook
- Body response data of the webhook script on your server
- HTTP response code of the webhook script on your server
- Date/time when the webhook was added/triggered
- Whether or not the webhook was processed
- Date/time when the webhook was processed

4.3 Accessing logs

You parties are able to access the log mentioned above by sending the following request to OWS, where **[sessionid]** is the session ID your session you received by providing your API key and **[numberofresults]** is the desired number of results (optional, default is '10'):

```
{
  "jsonrpc": "2.0",
  "method": "Otys.Services.WebhooksService.getLogs",
  "params": [
      "[sessionid]",
      "[numberofresults]"
  ],
  "id": 1
}
```

If a valid session ID has been provided, this will result in the following response:

```
{
  "result": [
    [
      {
        "id": "[id]",
        "klantid": "[klantid]",
        "userid": "[userid]",
        "webhook_url": "[webhookurl]",
        "request_data": "[requestdata]",
        "response_data": "[responsedata]",
        "response_code": "[responsecode]",
        "added": "[addeddatetime]",
        "is_processed": "[processed]",
        "dt_processed": "[processeddatetime]"
      }
    ]
  ],
  "id": 1
}
```

Please note the result object consists a recursive array. So if there are five webhook log items you will see five sets of log fields.

The response consists of the following fields per log item:

- **[id]**: ID of the log item.
- **[klantid]**: Client ID of the user which executed the OWS request that triggered the webhook.
- **[userid]**: User ID of the user which executed the OWS request that triggered the webhook.
- **[webhookurl]**: Webhook URL that has been triggered.
- **[requestdata]**: The OWS request & OWS response that triggered the webhook and which was posted to the webhook URL. Please note that out of security reasons we will remove some data like login requests & session IDs.
- **[responsedata]**: The response the webhook URL gave on the post in the body. This way external parties are able to add debug information to the body of the URL which processes the webhook and are able to see this debug information in the log on the OTYS side.
- **[responsecode]**: The response code the webhook URL gave on the post of the body (normally a '200', but can for example also be a '404' or a '500').
- **[addeddatetime]**: Date & time when the item has been added for processing. Since processing is triggered by RabbitMQ, execution date & time (which is not logged) will be very shorlty after this date & time.
- **[processed]**: Indicator whether the item has been processed ('1') or not ('0'). Since processing is triggered by RabbitMQ, all records will normally be set to '1'.
- **[processeddatetime]**: Date & time when the item has been actually processed. Since processing is triggered by RabbitMQ, process date & time will normally be very shortly after previously mentioned added time.