Recruiting Technology

Let's seize opportunities.

Together

Manual OTYS web services (OWS)

Version 1.2

A manual for external parties integrating with OTYS
to exchange (retrieve, add, update & delete)
data using the OTYS web services.

17-05-2023

Version management

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

Version	Date	Author	Description
1.0	20-05-2022 - 03-06-2022	Bastiaan Brans	Created initial version of document, including three chapters: 'General' (chapter 1), 'Basic requests' (chapter 2) & 'File management' (chapter 3).
1.1	15-07-2022	Bastiaan Brans	Updated 'OTYS Recruiting Technology' (paragraph 1.1) based upon current situation. Updated 'File management' (chapter 3) so that the endpoint is consistent with the endpoint mentioned in chapter 2.
1.2	17-05-2023	Bastiaan Brans	Added information to force an OWS request to be executed in a specific content language in 'Content languages' (paragraph 2.6).

Contents

1. General

In this chapter we will provide you some generic information about OTYS, the OTYS Go! Partner framework and the OTYS web services (OWS).

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 (agencies, corporate recruitment & job boards).

The most important elements of the OTYS solution are:

- OTYS Go!
  This is a browser based software solution in which the main users (for example recruiters) of the client work on a daily basis. Each client has a client specific URL from which they access OTYS Go! (https://clientprefix.otysapp.com). Users then login using either their OTYS username & password or using an external SSO option (for example Azure). After they are logged in, they will have access to the modules & functionalities based upon the permissions assigned to their account.
- Website
  In most cases the client also has one or multiple websites linked to their system. This is normally a website which is publicly available (so that the company can for example use it to show & tell more about their company and publish their jobs), with some portals behind a login (for candidates, customers, hiring managers and/or suppliers). This website will interact with the OTYS system and can do so in several ways (for example the main website build, hosted & maintained by a third party and portals build, hosted & maintained by OTYS).

1.2 OTYS Go! Partner Framework

OTYS believes it should focus on the elements they excel in and that their focus should be on just that. Other parties are able to do other things better and OTYS believes that their clients are best helped by having a 'best of breed'-solution using partners. To support this 'best of breed'-solution for both clients & partners, OTYS created the 'OTYS Go! Partner framework' which allows partners to easily connect to the OTYS solution and provide users a nice (seamless) experiences.

The OTYS Go! Partner framework consists of the following elements:

1. 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.
2. OTYS Go! Partner OWS Webhooks
   If an external party connects to OTYS using OWS, there is always some kind of 'trigger' to exchange data. If this trigger should be based upon a user action in OTYS Go! (for example a user changing a status) , it is possible to create OWS webhooks. If configured, the partner can receive an 'active signal' that a user is taking the action. This way as a partner you are able to 'instantly' start a process if for example a user changes a status in the OTYS Go! system.

3. OTYS Go! Partner SSO
   If an OTYS Go! user interacts with a partner functionality, it is probably a wise thing to add some kind of authentication. Larger clients will probably have an Active Directory (such as Microsoft Azure) set up to make user management & authentication work smoothly. Smaller clients will probably not have set this up, so for those we created an OTYS Go! Partner SSO functionality, which allows partners to automatically sync users with OTYS Go! users and to easily authenticate users using Single Sign On (with OTYS being the 'Active Directory').
4. OTYS Go! Partner widgets
   In some cases a partner wants to be able to show data from their own system to users of OTYS Go! within the context of OTYS Go! (for example show additional information you have about a candidate within an OTYS Go! candidate detail). You might even want to add some 'interaction' in this context (for example a button to click or a form to be submitted). For these kinds of scenarios we have created OTYS Go! Partner widgets, which allows partners to provide iframe-ish content which can be loaded directly into the right context in OTYS Go!
5. OTYS Go! Partner custom actions
   The above mentioned OTYS Go! Partner widgets can be of great added value if you want to show some information within the context of OTYS Go!, however if you have more complex ('full blown') interfaces or simply want to 'trigger a specific action', the widgets might not be the best choice. For these kinds of interactions we have created 'OTYS Go! Partner custom actions' which allows partners to create custom buttons which can be added on various places in the OTYS Go! system (for example in a candidate detail). Clicking this button will open a URL on your side (with for example the candidate OUID as a parameter) in the current browser tab, a new browser tab, a modal, an overlay or on the background.

1.2 OTYS web services

OTYS has very extensive web services (called 'OTYS web services' or simply 'OWS') 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. OWS is available in both JSON-RPC format as XML-RPC format.

All actions users take through the user interface (OTYS Go!) are done by OWS requests using the JSON-RPC service, 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.

OWS can be used for all kinds of database interactions, including creating websites for OTYS clients that 'interact' with the OTYS website. This way an external web builder is for example able to retrieve all published or specific vacancies to include on the website.

Most requests done through OWS are done for retrieving, adding, updating or deleting records (for example a request to retrieve published vacancies or a request to create a new candidate). It is also possible to retrieve & upload files (for example a candidates CV) and link them to specific database records (for example a candidate). This process of retrieving & uploading files and connecting them to the correct database records works a bit 'different' from other OWS methods and is therefore specifically mentioned in a separate chapter.

1.4 About this document

This document describes how you can use the OTYS web services (OWS) to exchange (retrieve, add, update & delete) data. It is divided into the following three chapters:

1. General
   In this chapter we will provide you some generic information about OTYS, the OTYS Go! Partner framework and the OTYS web services (OWS).
2. Basic requests
   In this chapter we will go through some 'OWS basics' like the different versions we have, how to authenticate and how some basic requests (like retrieving, adding, updating & deleting data) look like.
3. File management
   As mentioned above the process of retrieving & uploading files and connecting them to the correct database records works a bit 'different' from other OWS methods and is therefore specifically mentioned in this chapter.

At a later stage we will add a fourth chapter, which will provide you with some example use cases with relevant requests.

If you have feedback about this document, we would love to hear it from us. Please send us an email at partners@otys.com and provide us your feedback.

2. Basic requests

In this chapter we will go through some 'OWS basics' like the different versions we have, how to authenticate and how some basic requests (like retrieving, adding, updating & deleting data) look like. Based upon this chapter you should be 'good to go' to start building your integration with OTYS using our OTYS web services.

2.1 Needed credentials

To be able to authenticate, you will of course need credentials. To be able to integrate with OTYS using OWS, you will need the following credentials:

- API key
  The API key is needed to obtain a session token (see paragraph 2.3) which is needed to do follow up requests to exchange data.
- Username & password
  The API key is linked to a specific user in the OTYS system (and therefore has the same permissions as this user). This user has a username & password and this username/password can be for example used to retrieve a list of field definitions or log into OTYS Go!
- OTYS Go! URL
  Each OTYS client has a client specific domain (https://[clientcode].otysapp.com) which is used to login to the OTYS Go! application. You can use this URL and the above mentioned username & password log into the OTYS Go! GUI to get an understanding of how the OTYS system works 'from a user perspective' (and intercept example OWS requests).

Normally a client will request the credentials mentioned above from OTYS, OTYS will send the credentials to the client and the client can forward the credentials to you.

2.2 Endpoints

The OTYS web services (OWS) is available in two protocols (JSON-RPC & XML-RPC). Each protocol has its own endpoint and is used for all normal data exchange (retrieving, adding, updating & deleting records like 'candidates', 'vacancies' & 'customers'). Specifically for uploading & downloading files we have a third endpoint (which you will have to use when 'working with files' whether you use the JSON-RPC or XML-RPC protocol). The endpoints for these services are:

- JSON-RPC service
  For requests & responses in JSON format, endpoint: https://ows.otys.nl/jservice.php
- XML-RPC service
  For requests & responses in XML format, endpoint: https://ows.otys.nl/service.php
- File service
  For multipart form-data requests (for uploading files) or HTTP requests (for downloading files), endpoint: https://ows.otys.nl/fileService.php

We strongly advise you to use the JSON-RPC version; since it is the version both most partners and ourselves prefer (OTYS Go! uses JSON-RPC). Therefore all example requests in this manual, example requests intercepted from OTYS Go! and most other examples we have are in JSON format. If you have good reasons to choose XML-RPC, you can of course use the XML-RPC version; however keep the above in mind.

2.3 Authentication

As mentioned in paragraph 2.1 you should have received an API key. Using this API key you can 'prove you are good people' and retrieve a 'session token'. Using this session token you can do follow up requests (such as retrieving vacancies & creating candidates).

2.3.1 Retrieve session token

So it is time for your very first OWS request. You can retrieve a session token by sending the following request to endpoint https://ows.otys.nl/jservice.php (which is also the endpoint for all other example requests mentioned in this chapter):

```
{
	"jsonrpc": "2.0",
	"method": "loginByUid",
	"params": [
		"[apikey]"
	],
	"id": 1
}
```

In the request above:

[apikey] Is the API key provided by OTYS.

If you have provided a valid API key, you will get the following response:

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

In the response above:

[sessiontoken] Is the session token of the created session which you can use in follow up requests.

2.3.2 Check session

Now you have a session token, it is kind of nice to use it. A very simple way to do so is to 'check' your session token. You will provide a session token and we will tell you if it is valid and -if it is- provide you information about the user & client connected to the session. You can check your session by sending the following request:

```
{
	"jsonrpc": "2.0",
	"method": "check",
	"params": [
		"[sessiontoken]"
	],
	"id": 1
}
```

In the request above:

[sessiontoken] Is the session token previously retrieved.

If you have provided a valid session token, you will get quite an extensive response with quite a lot of field about the user & client connected to your current session. To keep things simple, we will show you a simplified request below with the most important fields of this response:

```
{
    "result": {
        "clientId": [clientid],
        "client": "[clientname]",
        "id": [userid],
        "username": "[username]",
        "email": "[useremail]",
        "contentLanguages": [
            "[contentlanguage]"
        ],
        "defaultContentLanguage": "[defaultcontentlanguage]"
    },
    "id": 1
}
```

In the response above:

[clientid] This is the numeric ID of the OTYS client (for example '1234'). If you integrate for multiple OTYS clients, this is probably the best field to distinguish the individual clients.

[clientname] This is the name of the OTYS client (for example 'Recruitment Heroes'). If you are a human being, this is probably the easiest way to identify the client.

[userid] This is the numeric ID of the OTYS user (for example '12345'). If you integrate for multiple OTYS users, this is probably the best field to distinguish the individual users.

[username] This is the username of the OTYS user connected to the session token (for example 'partnerxyz.recruitmentheroes'. If you want to log into OTYS Go! as the same user, this is probably the best way to identify the linked user.

[useremail] This is the (main) email address of the OTYS user (for example 'partner@recruitmentheroes.nl'). If you are a human being, this is probably the easiest way to identify the user.

[contentlanguage] The OTYS system allows users to store some data in multiple languages. This way for example one vacancies can be created in multiple languages (so that it is technically one vacancy with multiple language versions). This shows the available content languages for this client/user in ISO 639-1 format. Please note this is an object which can contain one or multiple content languages

[defaultlanguage] This is the default content language of the user. If the user would log into OTYS Go!, this would be the content language which is 'by default selected'. A user needs to take a specific action to 'switch' to a different content language.

2.3.3 Log out

If you are finished using an software application (for example OTYS Go!) and you are a 'good citizen'; you will of course log out of the application. If you are finished using OWS (and you are a 'good citizen' 😊) it is of course good to log out of OWS / kill your session. You can do so, by sending the following request:

```
{
	"jsonrpc": "2.0",
	"method": "logout",
	"params": [
		"[sessiontoken]"
	],
	"id": 1
}
```

In the request above:

[sessiontoken] Is the session token of the created session which you can use in follow up requests.

If you have provided a valid session token, you will get the following response:

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

The session is now not valid anymore and you are 'logged out'. If you send another request with the same session token (for example logging out for a second time, you will get the following response):

```
{
	"error": {
		"message": "Access denied; please login first",
		"code": 0
	},
	"id": 1
}
```

2.3.4 Session management

To make sure session handling is implemented correctly in your integration, as mentioned above:

- You can provide your API key and retrieve a session token using service call 'loginByUid';
- You can check your current session using service call 'check';
- You can log out / kill your session using service call 'logout'.

Please make sure this logic is in your application. As the word says a 'session' is considered to be a 'session'. We have had partners in the past which (initially) manually created a session token and stored the session token in their application which at some point can result in the above mentioned 'Access denied; please login first'-message. Sessions can at some point not be valid anymore (for

example by another process ending the session or by OTYS automatically killing a session after a period of inactivity). Additionally, some things (for example the 'current content language') can be linked to a session.

The 'best way' to integrate with OTYS is to have a single process (which uses a 'queue') to handle all OWS requests for a specific API key. This way you can (a) start the session, (b) execute follow up requests in the desired order and (c) end the session when done.

2.4 Data service & field definitions

We have an extensive list of available data services & field definitions. This list might feel a bit overwhelming the first time you open it, but fortunately we have an easy way to pinpoint specific data services & fields (more about that later in this chapter).

However, one way or the other, it is good to have an overview of these available data services & field definitions. If you for example have some basic request (for example to create a new candidate) and are looking for some specific field which might not be in that request, you can check our online list of data services and fields that are part of OWS.

You can access this list by opening the following URL in your browser:
https://ows.otys.nl/info/

After opening the URL, you are prompted to enter a username & password. Here you are able to enter the username & password received from OTYS. Once you have provided that, you will see a list of over 400 data services (for example 'Otys.Services.CandidateService' for the data service which allows you to exchange the main candidate data). If you click on a data service, you will see the field definitions of that specific data service (so for example the one of the main candidate data).

For every field you will see the following information:

- Name
  This is the name of the field which you can use when defining which fields you want to be returned ('what'), which records you want to be returned ('condition') and how you want records to be ordered ('sort').
- Type
  This is the data type of the field (for example 'string', 'boolean' or 'datetime') so that you know in which data format you should send the information and what you can expect when receiving data.
- Listable
  This checkbox will be 'checked' if you can retrieve the field when getting a list of items using the getList(Ex)-method. If a filed is not listable, you will need to do an additional request (probably a getDetail) to retrieve the additional fields.
- Filterable
  This checkbox will be 'checked' if you can filter based upon this field using the getList(Ex)-method as a 'condition'.
- Sortable
  This checkbox will be 'checked' if you can order based upon this field using the getList(Ex)-method as a 'sort'.
- Updatable
  This checkbox will be 'checked' if you can update this field. Most fields are updatable,

however there are some fields which are 'view only' or require special methods to update the fields.

- Obligatory
  This checkbox will be 'checked' if the field is mandatory (and should be filles, so not 'null') when creating new records using the add-method.
- Deletable
  This checkbox will be 'checked' if you can 'delete' the field'. This is normally only the case for 'collection' fields. It is for example not possible to 'delete' the candidates first name (you can of course update it so that it becomes empty), but you can delete one of the candidates work experience items.
- Insertable
  This checkbox will be 'checked' if you can define this field when inserting a new record using the add-method. If a field is not insertable but is updatable (most times for technical reasons because something first needs to be 'there' before we can update it) you can insert the record without this field and then do another request to update the created record so that the field is filled for the newly created record.
- Has options
  This checkbox will be 'checked', the field will have specific options. You can therefore not simply send 'some data' but should specific option (normally an ID) and will get that returned as well. In paragraph 2.7 you will find how you can retrieve the options of these fields.
- Not empty
  This checkbox will be 'checked' if the field cannot be included 'empty' when creating new records using the add-method or updating existing records using the update-method (however you can exclude them from the request as a whole).
- Has hints
  This is a deprecated functionality and not relevant for partner integrations.
- In list by default
  If you retrieve a list of items, you should normally define which fields you would like to receive using the 'what'-object. If you do not define which fields you want to get returned, we will return the fields which are set to be shown in lists by default. However, we strongly (strongly!) advice you to define which fields you want to have returned since it will make the integration faster (only retrieving data you actually need) and more stable (if there are changes to the default fields this cannot break your current integration).
- Data type comment
  In this field we can give some more information about the data type of the field (for example if it is a 'datetime' field, the format of the datetime).
- Comment
  In this field we can give you some additional comments about the field which can be 'technical things to keep in mind' or more functional things.

Please note that fields with type 'collection' & 'container' are by default collapsed in the field definition list. You will see an arrow-icon on the left side of the field. Clicking on this icon will expand the field, showing the underlaying fields (which can also include collection/container fields which will then again by default be collapsed). So if you are a bit overwhelmed by opening the field definition list of for example our CandidateService, you will probably be even more overwhelmed after expanding all collections & containers.

2.5 Basic methods

2.5.1 Introduction

Now you have seen the extensive list of available data services & fields (and might be a bit overwhelmed by them), let's make things a bit more simple by going through some basic methods.

Although data services can have many (very specific) methods, there are some basic methods which are supported by most data services and which you will use in most use cases. These methods are:

- getList(Ex)
  To retrieve a list of record (for example all candidates with a specific main status).
- add
  To create a new record (for example create a new candidate).
- update
  To update an existing record (for example update an existing candidate).
- delete
  To delete an existing record (for example delete an existing candidate).

You will use this method in combination with the relevant data service. So if you for example:

- Want to list candidates, you will use 'CandidateService.getList(Ex)';
- Want to add a vacancy you will use 'VacancyService.update';
- Want to update a customer / relation you will use 'RelationService.update';
- Want to delete a customer / relation contact person you will use 'RelationContactService.delete'.

We will use the CandidateService (so listing, adding, updating & deleting candidates) as an example for some basic requests.

2.5.2 Search & retrieve a list of records

As you can see, this is the only method where we placed something in between parentheses, because there are two methods to retrieve a list of records: 'getList' and 'getListEx'.

The methods 'getList' & 'getListEx' are both very similar, however the method 'getListEx' has some additional possibilities including getting a total count. This way you can for example retrieve candidates with a specific main status ordered by entry date descending (so from new to old), retrieve the first 100 items AND get a total count. This way you will only retrieve the first 100 candidates, but also get a total count back of the total amount of candidates with that main status; allowing you to do the right amount of additional requests to retrieve all items OR to create a nice page navigation in your application showing live data.

A basic getListEx request retrieving data will be as follows:

```
{
	"jsonrpc": "2.0",
	"method": "Otys.Services.CandidateService.getListEx",
	"params": [
		"[sessiontoken]",
		{
			"what": {
				"uid": 1,
				"Person": {
					"firstName": 1,
					"lastName": 1
				}
			},
			"condition": {
				"type": "AND",
				"items": [
					{
						"type": "COND",
						"field": "userId",
						"op": "EQ",
						"param": "[userid]"
					},
					{
						"type": "COND",
						"field": "statusId",
						"op": "NE",
						"param": "[statusid]"
					}
				]
			},
			"sort": {
				"Person.lastName": "ASC",
				"Person.firstName": "ASC"
			},
			"limit": 100,
			"offset": 0,
			"getTotalCount": 1
		}
	],
	"id": 1
}
```

As you can see, this request consists of the following basic structure:

- Method
  It will define the method (in this case 'Otys.Services.CandidateService.getListEx'), so that we know you are requesting a list of candidates.
- Authentication
  It will then pass the session token you have received during the authentication process, so that we know you are 'good people' (and can provide the correct records).
- What
  It will then pass a 'what' object to define which fields you want to returned for every record

(in this case UID, first name & last name). To make integrations as fast & stable as possible, please make sure to only retrieve fields that you actually need.

- Condition
  It will then pass a 'condition' object to define conditions, so that we know which records to return (in this case all candidates that have a specific userId/owner and that do NOT have a specific statusId/main status). If you do not provide this conditions object, we will return all available records. As you can see it is possible to combine multiple conditions in AND/OR structures. The available operators ('op') are (please note not all operators are available for all field types:
  - EQ: field is equal to parameter
  - NE: field is not equal to parameter
  - GE: field is greater or equal the parameter
  - GT: field is greater than parameter
  - LE: field is less or equal than parameter
  - LT: field is less than parameter
  - T: boolean field is true
  - F: boolean field is false
  - LIKE: matching a pattern
  - NLIKE: not matching a pattern
- Ordering
  If will then pass a 'sort' object to define sorting/ordering (in this case on candidate last name & first name ascending), so that we know in which order you would like to receive the results. Of you do not provide this sort object, we will return the records in some predefined order (normally ID, but can differ per service). As you can see it is possible to order on multiple fields, with per field the following ordering options:
  - ASC: field is ordered ascending (from A to Z, from small to large, from old to new)
  - DESC: field is ordered descending (from Z to A, from large to small, from new to old)
- Limiting
  It will finally define which records it will return (in this case the first 100 records including a total count), so that we know which records from the full results you want to receive. Please note that most services provide a maximum of 200 results per request (so setting 'limit' higher than '200' will still return 200 results). If in the response of the request above you receive the first 100 records and a total count of 350, you know you will need to make three additional requests (offsets 100, 200 & 300) to retrieve all 350 candidates.

The request above will result in a response as follows:

```
{
	"result": {
		"listOutput": [
			{
				"uid": "[uid]",
				"Person": {
					"firstName": "[firstname]",
					"lastName": "[lastname]"
				}
			}
		],
		"searchExtras": [],
		"totalCount": [totalcount]
	},
	"id": 1
}
```

The items in the listOutput object are of course recursive (and can contain 0, 1 or multiple records), with as a maximum the set 'limit'. The **[totalcount]** will show you the total count, allowing you to do the right amount of additional requests to retrieve all items OR to create a nice page navigation in your application showing live data.

2.5.3 Retrieving a specific record

A basic update request to retrieve a specific existing record will be as follows:

```
{
	"jsonrpc": "2.0",
	"method": "Otys.Services.CandidateService.getDetail",
	"params": [
		"[sessiontoken]",
		"[candidateouid]",
		{
			"Person": {
				"firstName": 1,
				"lastName": 1,
			},
			"statusId": 1,
			"userId": 1
		}
	],
	"id": 1
}
```

As you can see, this request consists of the following basic structure:

- Method
  It will define the method (in this case 'Otys.Services.CandidateService.getDetail), so that we know you want to retrieve a specific existing record.
- Authentication
  It will then pass the session token you have received during the authentication process, so that we know you are 'good people'.

- Record ID
  Since you are requesting a specific existing record, we want to know which record you want to retrieve. In this case (for retrieving a candidate), you will need to provide the candidate OUID.
- Fields
  It will then pass the fields which you want to retrieve. Normally you would only provide us fields that you actually want to be changed (fields that are not provided will stay intact).

The request above will result in a response as follows:

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

2.5.4 Adding new records

A basic add request to add a new record will be as follows:

```
{
      "jsonrpc": "2.0",
      "method": "Otys.Services.CandidateService.add",
      "params": [
            "[sessiontoken]",
            {
                  "Person": {
                        "firstName": "[firstname]",
                        "lastName": "[lastname]"
                  },
                  "statusId": [statusid],
                  "userId": [userid]
            }
      ],
      "id": 1
}
```

As you can see, this request consists of the following basic structure:

- Method
  It will define the method (in this case 'Otys.Services.CandidateService.add'), so that we know you are creating a new candidate.
- Authentication
  It will then pass the session token you have received during the authentication process, so that we know you are 'good people'. In most cases if you create a new record and do not assign an owner of the newly created record (in this case 'userId') we will make the user linked to your API key the owner.

- Fields
  It will then pass the fields which are needed to create the record, so that we 'fill in all fields' as desired.

The request above will result in a response as follows:

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

In most cases when creating a new record, we will provide you with the ID of the created record in the response. This way if you need to do follow up actions (for example link the created candidate to a vacancy) or want to store it in your database (allowing you to sync items) you have the record ID to do so.

2.5.5 Updating existing records

A basic update request to update an existing record will be as follows:

```
{
	"jsonrpc": "2.0",
	"method": "Otys.Services.CandidateService.update",
	"params": [
		"[sessiontoken]",
		"[candidateouid]",
		{
			"Person": {
				"firstName": "[firstname]",
				"lastName": "[lastname]"
			},
			"statusId": [statusid],
			"userId": [userid]
		}
	],
	"id": 1
}
```

As you can see, this request consists of the following basic structure:

- Method
  It will define the method (in this case 'Otys.Services.CandidateService.add'), so that we know you are creating a new candidate.
- Authentication
  It will then pass the session token you have received during the authentication process, so that we know you are 'good people'. In most cases if you create a new record and do not assign an owner of the newly created record (in this case 'userId') we will make the user linked to your API key the owner.
- Record ID
  Since you are updating an existing record, we want to know which record you want us to update. In this case (for updating a candidate), you will need to provide the candidate OUID.

- Fields
  It will then pass the fields which are needed to update the record, so that we 'update all fields' as desired. Normally you would only provide us fields that you actually want to be changed (fields that are not provided will stay intact).

The request above will result in a response as follows:

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

2.5.6 Deleting records

A basic update request to update an existing record will be as follows:

```
{
      "jsonrpc": "2.0",
      "method": "Otys.Services.CandidateService.delete",
      "params": [
            "[sessiontoken]",
            "[candidateouid]"
      ],
      "id": 1
}
```

As you can see, this request consists of the following basic structure:

- Method
  It will define the method (in this case 'Otys.Services.CandidateService.delete'), so that we know you are deleting a candidate.
- Authentication
  It will then pass the session token you have received during the authentication process, so that we know you are 'good people'.
- Record ID
  Since you are deleting a record, we want to know which record you want us to delete. In this case (for deleting a candidate), you will need to provide the candidate OUID.

The request above will result in a response as follows:

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

2.6 Content languages

The OTYS system is 'multilingual' in two different ways:

- The application is available in multiple 'application languages'. So users of OTYS Go! are able to see the same contents, however the application language (so the 'buttons', 'labels' and those kinds of elements) can be different.
- Various entities & many fields are available in in multiple 'content languages'. If you for example have one vacancy you want have published in both Dutch & French; you are able to store various fields (job title, job description, job requirements, etc) in these multiple languages. This way the OTYS system knows it is the same record (candidates applying to both the Dutch & French language version will appear in the same 'pipeline' of the same vacancy.

From an 'API perspective' the application language is not really interesting (except maybe if you open OTYS Go! and seeing all kinds of Dutch labels, knowing that you can 'switch' it to English); however the content language is important. Many clients use online one content language, however if you are building an integration for a client who uses multiple content languages it is of course important to retrieve, create & update the records in the correct language.

Set current content language for follow up requests

You do NOT need to define the content language in every request. Instead, you can define your 'current content language' in a separate request. All operations after this request are then executed in that content language (until you tell us to switch to another language). For this purpose, most services have a 'setLanguage' method; which you can use if you want to retrieve, add or update language specific contents.

A basic update request to update the current content language will be as follows:

```
{
	"jsonrpc": "2.0",
	"method": "Otys.Services.[dataservice].setLanguage",
	"params": [
		"[sessiontoken]",
		"[language]"
	],
	"id": 1
}
```

As you can see, this request consists of the following basic structure:

- Method
  It will define the method, so that we know you are changing the content language. You will change **[dataservice]** into the applicable data service. If you for example want to set the content language for the 'VacancyService' data service, the method will be 'Otys.Services.VacancyService.setLanguage'.
- Authentication
  It will then pass the session token you have received during the authentication process, so that we know you are 'good people' AND we are changing the current content language of the user linked to your API key (so that it does not change anything for other users).

- Language
  Finally we of course want to know the language you want to set. Languages are expected in ISO 639-1 format. Supported content languages currently are:
  - cs: Czech
  - de: German
  - en: English
  - es: Spanish
  - fr: French
  - nl: Dutch, Flemish
  - us: English (US)

The request above will result in a response as follows:

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

Define specific language for specific request

Alternatively, it is also possible to use the 'Otys.Services.LanguageService.wrapRequest'-method to 'wrap' your request and force the content language in which the request will be executed. This will 'overrule' the currently set content language.

A basic request which retrieves the vacancy title in a specific language will be as follows:

```
{
    "jsonrpc": "2.0",
    "method": "Otys.Services.LanguageService.wrapRequest",
    "params": [
        "[language]",
        {
            "jsonrpc": "2.0",
            "method": "Otys.Services.VacancyService.getDetail",
            "params": [
                "[sessiontoken]",
                "[vacancyouid]",
                {
                    "title": 1
                }
            ],
            "id": 1
        }
    ],
    "id": 1
}
```

As you can see, this request consists of the following basic structure:

- Wrapper Otys.Services.LanguageService.wrapRequest with defined language
  Your original request should be wrapped in this Otys.Services.LanguageService.wrapRequest and you should define the **[language]** in which you expect the request to be executed. Languages are expected in ISO 639-1 format. Supported content languages currently are:

- cs: Czech
- de: German
- en: English
- es: Spanish
- fr: French
- nl: Dutch, Flemish
- us: English (US)

- Original request
  It will then include the original request. In this example it is a request of method 'Otys.Services.VacancyService.getDetail' to retrieve details of a specific vacancy, passing the **[sessiontoken]** and **[vacancyouid]** and requesting the field 'title'.

The request above will result in a response as follows:

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

2.7 Option lists

As mentioned in paragraph 2.7, some fields (normally pick lists in the application) include 'option lists'. For these fields it is of course important to know the available options, so that you can provide us with the correct contents (provide the correct ID when updating a candidate status) or so that you can interpret the provided contents better (know the name of the user when retrieving the owner ID of a candidate).

We have a generic method 'getOptionLists' which can be used in most data services. A getOptionLists request will be as follows:

```
{
      "jsonrpc": "2.0",
      "method": "Otys.Services.[dataservice].getOptionLists",
      "params": [
            "[sessiontoken]",
            [
                  "[field]",
                  "[field]"
            ]
      ],
      "id": 1
}
```

As you can see, this request consists of the following basic structure:

- Method
  It will define the method, so that we know you are retrieving the options. You will change **[dataservice]** into the applicable data service. If you for example want to retrieve options for

the 'CandidateService' data service, the method will be 'Otys.Services.CandidateService.getOptionLists'.

- Authentication
  It will then pass the session token you have received during the authentication process, so that we know you are 'good people'.
- Fields
  You can then pass one or multiple field names that 'have options' to retrieve the options of that list.

The request above will result in a response as follows:

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

Although the structure mentioned above might differ a bit for specific option lists, you will most probably get the point. You will receive the requested lists including their IDs & values.

2.8 Retrieving specific requests using OTYS Go!

As mentioned before, the OTYS Go! application recruiters use fully relies on the OTYS web service (OWS). If a user is taking specific actions in OTYS Go! (for example retrieving vacancies or creating a candidate), OWS requests are executed 'on the background'.

Because the OTYS Go! system was build 'API first', our web services are (as you will have experienced now) extensive. Very extensive. Everything a user is able to do 'database wise' from the OTYS Go! system is available in OWS. On the one hand this creates a lot of possibilities for partners integrating using OWS. On the other hand it makes it more difficult to find the correct request. Of course we do have the previously mentioned (long) list with available data services and fields that can be used in these data services, however there is an 'easier way'.

A very 'easy way' of retrieving a very specific request is by executing specific actions in the OTYS Go! application (for example updating the candidates UTM tags) and then intercept the request. This way you immediately know what the exact request is of that specific user interaction, which you can then adjust if you like.

You can use any HTTP web debugging proxy method (like Fiddler, Charles or build in browser solution) you prefer. Below you will find a step-by-step instruction on how to intercept the request using Google Chrome's build in functionality (not because it is 'the best', but because you probably already have this on your computer):

1. Open Google Chrome;
2. Open the provided OTYS Go! URL (see paragraph 2.1);
3. Go to the interface in OTYS Go! just where you will execute the 'action', however don't execute the action yet. So if you for example want to know how to update specific information in a candidate record; go to the place where you can update the information, fill in the fields but do not hit the save-icon/button yet;
4. Click on F12;
5. Open tab 'Network';
6. Make sure the filter below it is not active (so it shows 'All' requests);
7. Now execute the action (for example hit the save-icon/button);
8. You will see one or (probably) multiple requests of 'jservice.php'. Click on the first request.
9. Open tab 'Payload'. You will see the payload of the request and can switch between a 'parsed' version (which is might be easy to inspect) and a 'source' version (which might be easy to copy/paste so that you can alter & try out the request).
10. If the payload mentioned above is not the desired request, you can open the next request to see if that is the one you need.

Please note that as mentioned above a single click in the OTYS system many times leads to multiple requests to OWS (for example first saving fields and then retrieving updated fields). If you are intercepting the requests mentioned above for the first time, it might take a bit longer (since you are really looking closely to every request). If you have done this a couple of times, you will become more convenient with this process (for example if you are looking for a request to update UTM tags, 'scan' the request for 'UTM' and find it more quickly).

3. File management

In this chapter we will explain from a 'conceptual point of view' how you can upload & download files using OWS and how you can connect them to a database records (for example a candidate). Additionally we will provide a step-by-step description on how you can execute these actions using Postman (https://www.postman.com/), an API platform used by many developers to easily test the working of an API from a user interface (before actually starting coding the integration).

3.1 Uploading files

We will start with the process of uploading files and binding them to the correct database record.

3.1.1 Concept

Before you can upload & bind a file using OWS, you will need to have the following:

- You will need a valid OWS session token. This is the session token you have received by doing a request using service call 'loginByUid' and is used by all follow up requests. Since we are assuming you are familiar with OWS, we expect you to have this session token already.
- You will need to know the entity (for example 'Candidates') and the record ID (for example the OUID of a specific candidate) you want to add the file to. Since we are assuming you were already able to do various OWS operations (for example create a new candidate), we expect you to have this data already.

Upload file

First step is to upload the file to our webservice and retrieve a OUID of the uploaded file. For this you will need to do a POST request using a multipart form-data request to the OWS endpoint https://ows.otys.nl/fileService.php (please note this is a different endpoint than your 'normal' OWS requests) with two key-value pairs:

- sessionId: With as value your valid OWS session token;
- file: With as value the file you wish to upload.

If your request is successful, you will get a response in JSON format containing META data of the file:

```
{
	"[filename]": {
		"ouid": "[fileouid]",
		"name": "[filename]",
		"mimeType": "[filemimetype]",
		"size": [filesize]
	}
}
```

As you can see, this response consists of the following fields:

- **[filename]**: The filename of the received file, which can be used for validation purposes;
- **[fileouid]**: The OUID of the uploaded file on our servers. You will need this OUID for binding the uploaded file to a specific database record (for example a candidate);
- **[filemimetype]**: The mime type of the received file, which can be used for validation purposes;
- **[filesize]**: The size of the received file, which can be used for validation purposes.

Binding file

Once you have your file OUID, you can bind/connect it to the desired database record. Although files are used in multiple places in OWS and there are many methods that use file OUIDs, most of the times a file needs to be added to the 'dossier' of a specific entity. We will use this as an example.

Almost all OTYS clients use 'document types' when uploading files to their dossier. It allows them to define 'what kind of file' they are adding to the dossier (for example a CV, motivation letter or contract) which can be used for filtering, reporting and other purposes. Since these 'document types' is a client specific option list, you will first need to retrieve the current options. For this you will need to do a POST request using a JSON request to the OWS endpoint https://ows.otys.nl/jservice.php (please note this is the endpoint you use for all your 'normal' OWS requests) which looks as follows:

```
{
	"jsonrpc": "2.0",
	"method": "Otys.Services.DocumentTypeService.getListEx",
	"params": [
		"[sessiontoken]",
		{
			"what": {
				"uid": 1,
				"name": 1,
				"forDocuments": 1,
				"forCandidates": 1,
				"forVacancies": 1,
				"forCustomers": 1,
				"forCustomerContacts": 1,
				"forTasks": 1,
				"forSales": 1,
				"forProjects": 1
			},
			"limit": 100,
			"offset": 0,
			"getTotalCount": true
		}
	],
	"id": 1
}
```

Please note service call 'DocumentTypeService.getList' also allows you to use some additional fields (the request above shows the most important & logical ones). You can see a complete overview of available fields on https://ows.otys.nl/info/detail.php?service=Otys.Services.DocumentTypeService (which requires you to login with your OTYS username & password).

If your request is successful, you will get a response in JSON format containing the current document types and the requested fields per document type, which looks as follows:

```
{
	"result": {
		"listOutput": [
			{
				"uid": "[uid]",
				"name": "[name]",
				"forDocuments": [forDocuments],
				"forCandidates": [forCandidates],
				"forCustomers": [forCustomers],
				"forCustomerContacts": [forCustomerContacts],
				"forVacancies": [forVacancies],
				"forTasks": [forTasks],
				"forSales": [forSales],
				"forProjects": [forProjects]
			}
		],
		"searchExtras": [],
		"totalCount": [totalCount]
	},
	"id": 1
}
```

As you can see, this response consists of the following fields:

- **[uid]**: This is the UID of the document type. You will need this UID when binding the file to the record later;
- **[name]**: This is the name of the document type. It allows you to distinguish the various document types 'as a human';
- **[forDocuments]**: Document types can be used for documents and notes. This is a boolean field if the document type is configured to be used for documents. You should only use document types when binding uploaded files to a dossier which have this field set to 'true';
- **[forCandidates]**: This indicates if the document type should be available for candidates. You should only use document types when binding uploaded files to a candidates dossier which have this field set to 'true';
- **[forCustomers]**: Similar as [forCandidates], but now for customers;
- **[forCustomerContacts]**: Similar as [forCandidates], but now for customer contacts;
- **[forVacancies]**: Similar as [forCandidates], but now for vacancies;
- **[forTasks]**: Similar as [forCandidates], but now for tasks;
- **[forSales]**: Similar as [forCandidates], but now for sales trajectories;
- **[forProjects]**: Similar as [forCandidates], but now for projects;
- **[totalCount]**: As you might have seen in the initial request, you are retrieving the first 100 document types (limit '100', offset '0'). Most clients will have (much) less than 100 document types. If the client however has over 100 document types you can either increase the 'limit' in your request or loop through results using this returned total count (so that if your first response includes a total count of 255, you can do two additional requests (offset '100' & offset '200') to retrieve all results.

Please note the example above only shows one document type, while in practice in most cases you will get multiple (recursive) items in your response.

If you want to assign a document type which is currently not available in the client, it is possible to add a new one. Although it is also possible to manage document types using OWS methods 'add',

'update' & 'delete'; since it is a 'one time thing' it is probably easier to do it from the OTYS Go! user interface, by taking the following steps:

1. Open OTYS Go!;
2. Login as a Key-user (user with additional permissions to change configuration);
3. Click on the users name on the top right and choose 'Client settings';
4. At 'Search setting number…' in panel on right type in 'GE91' followed by enter;
5. Open setting 'Dossier - Dossier types configuration' (GE91) in the list on the left;
6. You will see a user interface to add, update & delete document types.

Now you have the file OUID and the document type UID which you want to bind to the dossier of a specific record; you can now actually bind it to that record. For this you will need to do a another POST request using a JSON request to the OWS endpoint https://ows.otys.nl/jservice.php (please note this is the endpoint you use for all your 'normal' OWS requests) which looks as follows:

```
{
	"jsonrpc": "2.0",
	"method": "Otys.Services.AttachedDocumentsService.add",
	"params": [
		"[sessiontoken]",
		{
			"[entitykey]": "[recordouid]",
			"fileUid": "[fileuid]",
			"typeId": "[typeid]",
			"subject": "[subject]"
		}
	],
	"id": 1
}
```

As you can see, this request consists of the following fields:

- **[sessiontoken]**: The session token you use in every OWS request to authenticate;
- **[entitykey]**: The 'entity key' of the entity to which you want to upload the file to the dossier:
  - candidateUid: If you want to link the file to the dossier of a candidate;
  - relationUid: If you want to link the file to the dossier of a customer;
  - relationContactUid: If you want to link the file to the dossier of a customer contact;
  - vacancyUid: If you want to link the file to the dossier of a vacancy;
  - taskUid: If you want to link the file to the dossier of a task;
  - salesTrajectoryUid: If you want to link the file to the dossier of a sales trajectory;
  - projectUid: If you want to link the file to the dossier of a project;
- **[recordouid]**: The OUID of the actual record. So if at [entitykey] you have for example provided 'candidateUId', in this field you will provide the OUID of the candidate;
- **[fileuid]**: The file uid you have previously received after sending us the file;
- **[typeid]**: The document type ID as retrieved by the previous request;
- **[subject]**: The subject of the document as how it should show in the dossier (which can for example be the document name or a structured name like 'CV [candidatename]'.

Please note service call 'AttachedDocumentsService.add' (just like 'DocumentTypeService.getList') allows you to use some additional fields. You can see a complete overview of available fields on

https://ows.otys.nl/info/detail.php?service=Otys.Services. AttachedDocumentsService (which again requires you to login with your OTYS username & password).

If your request is successful, you will get a response in JSON format containing the current document types and the requested fields per document type, which looks as follows:

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

You can use the [dossierid] in follow up requests, however you will most probably not need that.

3.1.2 Examples using Postman

If you want to 'play around' with the process mentioned above, you can (for example) do so using Postman (https://www.postman.com/), an API platform used by many developers to easily test the working of an API from a user interface (before actually starting coding the integration).

Assuming you have Postman installed and are somewhat familiar to it, you can so by taking the following steps.

First let's upload a file to OWS:

1. Click on 'New request' (new tab) icon on top;
2. Change method-select (on left side of 'Enter request URL') to 'POST';
3. At 'Enter request URL' fill in 'https://ows.otys.nl/fileService.php';
4. Open tab 'Body' below it;
5. Select radio button 'form-data' below it;
6. At first key-field fill in 'sessionId';
7. At value on the right of it fill in your valid OWS session token;
8. At second key-field fill in 'file' and click anywhere out of the field to 'save' it;
9. Hover over the file-key you have just entered. A select will appear on the right. Change it from 'text' to 'file';
10. A 'select files' button will appear instead of the text value on the right. Click on it and select a file from your computer;
11. Click on button 'Send' on top right;
12. Response will be shown at the bottom. Change select 'HTML' on top of the response to 'JSON' to see a (nicer) JSON view of the response;
13. Copy the value of the 'ouid'-field and save it somewhere (for example in Notepad).

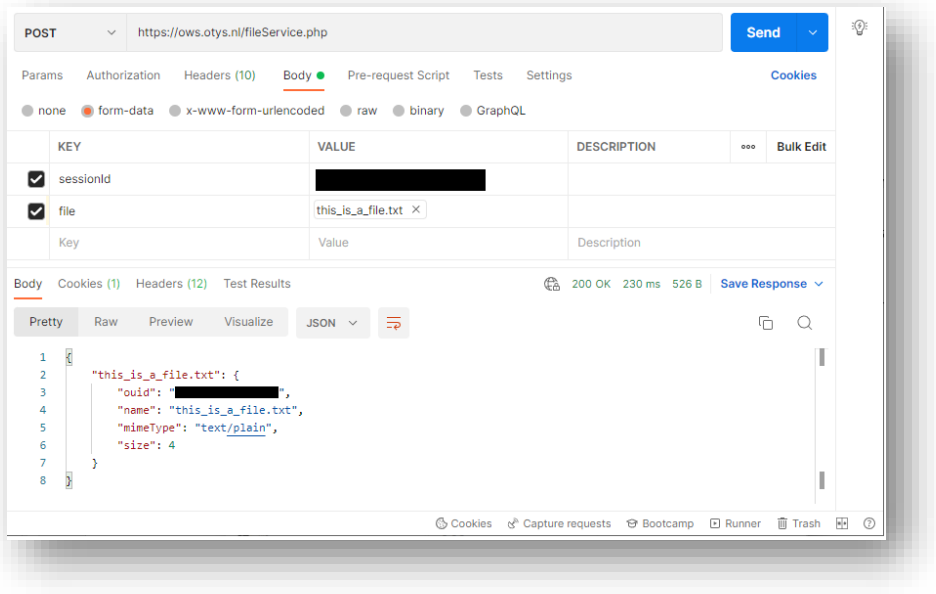

Image 3.1.2.1: Postman request uploading file to OWS

Now let's check the available document types:

14. Click on 'New request' (new tab) icon on top;
15. Change method-select (on left side of 'Enter request URL') to 'POST';
16. At 'Enter request URL' fill in 'https://ows.otys.nl/jservice.php';
17. Open tab 'Body' below it;
18. Select radio button 'raw' below it;
19. Paste the JSON of service call 'DocumentTypeService.getList' as shown in paragraph 3.2.1 in the text area below it;
20. Change the **[sessiontoken]** field into your valid OWS session token;
21. Click on button 'Send' on top right;
22. Response will be shown at the bottom. Pick a document type which has 'forDocuments' set to 'true' and the entity you would like to add the uploaded file to the dossier (for example 'forCandidates') to 'true, copy the value of the 'uid'-field and save it somewhere (for example in Notepad).

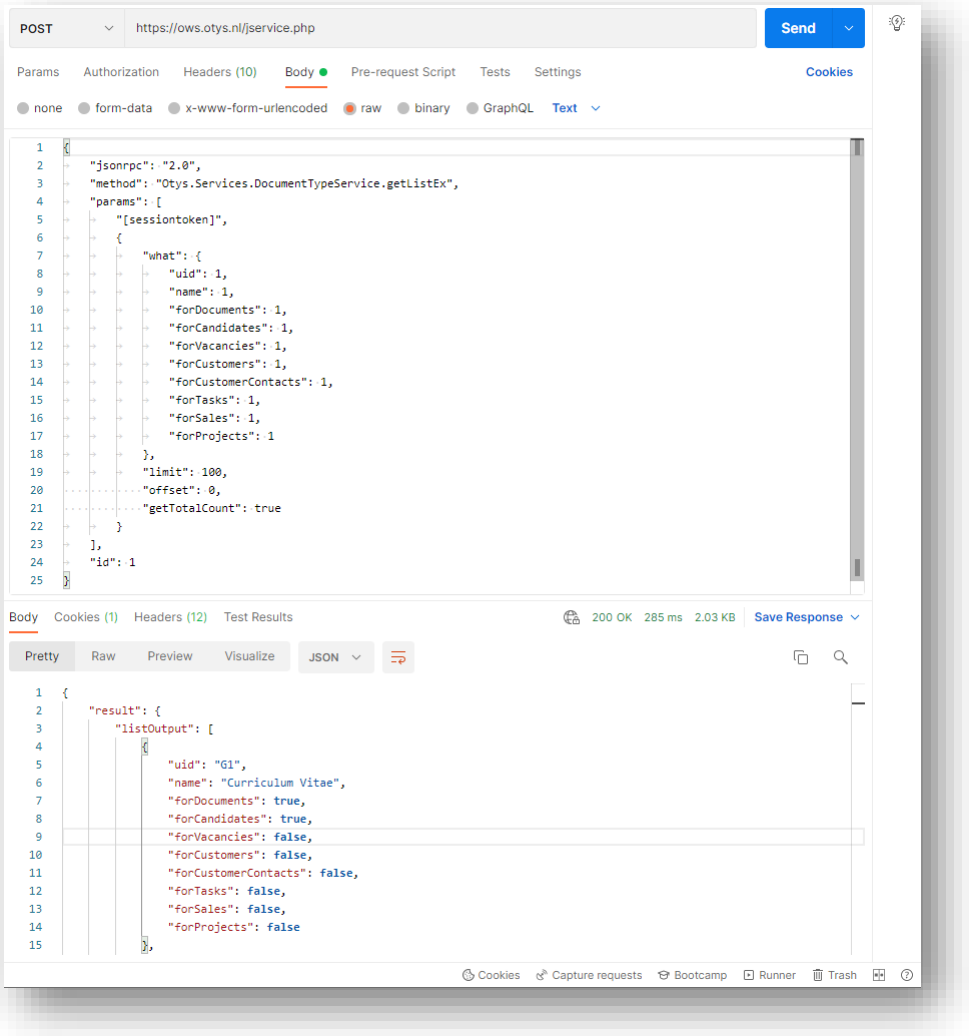

Image 3.1.2.2: Postman request retrieving document types

Now let's bind the actual file to a dossier as a document:

23. Click on 'New request' (new tab) icon on top;
24. Change method-select (on left side of 'Enter request URL') to 'POST';
25. At 'Enter request URL' fill in 'https://ows.otys.nl/jservice.php';
26. Open tab 'Body' below it;
27. Select radio button 'raw' below it;
28. Paste the JSON of service call 'AttachedDocumentsService.add' as shown in paragraph 2.2.1 in the text area below it;
29. Change the fields **[sessiontoken]**, **[entitykey]**, **[recordouid]**, **[fileuid]**, **[typeid]** & **[subject]** to the appropriate contents as mentioned in paragraph 2.2.1;
30. Click on button 'Send' on top right;
31. Response will be shown at the bottom with the ID of the created dossier item.

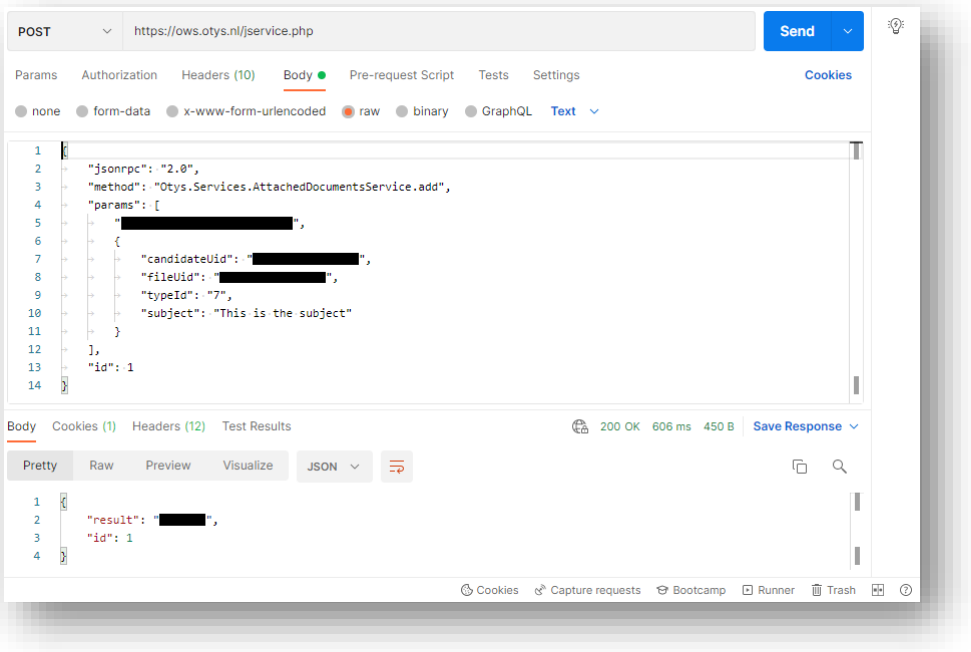

Image 3.1.2.3: Postman request binding file to dossier

3.2 Downloading files

Now you know how to upload files using OWS, we will continue with the process of downloading files from specific database records.

3.2.1 Concept

Before you can find & download a file using OWS, you will need to have the following:

- You will need a valid OWS session token. This is the session token you have received by doing a request using service call 'loginByUid' and is used by all follow up requests. Since we are assuming you are familiar with OWS, we expect you to have this session token already.
- You will need to know the entity (for example 'Candidates') and the record ID (for example the OUID of a specific candidate) for which you want to download the file. Since we are assuming you were already able to do various OWS operations (for example create a new candidate), we expect you to have this data again.

Find file

As mentioned in paragraph 3.2.1; files are used in multiple places in OWS and there are many methods that use file OUIDs, however most of the times a file is added to the 'dossier' of a specific entity. We will use this as an example again.

You can retrieve the documents which are attached to a dossier. For this you will need to do a POST request using a JSON request to the OWS endpoint https://ows.otys.nl/jservice.php (please note this is the endpoint you use for all your 'normal' OWS requests) which looks as follows:

```
{
	"jsonrpc": "2.0",
	"method": "Otys.Services.DossierService.getList",
	"params": [
		"[sessiontoken]",
		{
			"objectEntityId": [objectEntityId],
			"objectUid": "[objectUid]",
			"restrictClass": [
				"AttachedDocs"
			],
			"limit": 100,
			"offset": 0
		}
	],
	"id": 1
}
```

As you can see, this request consists of the following fields:

- **[sessiontoken]**: The session token you use in every OWS request to authenticate;
- **[objectEntityId]**: The 'entity ID' of the entity for which you want download the file from the dossier:
  - Candidates: Entity ID '2';
  - Customers: Entity ID '3';
  - Customer contacts: Entity ID '4';
  - Vacancies: Entity ID '1';
  - Tasks: Entity ID '28';

- Sales trajectories: Entity ID ‘48’;
- Projects: : Entity ID ‘158’;

- **[objectUid]**: The OUID of the actual record. So if at [objectEntityId] you have for example provided ‘2, in this field you will provide the OUID of the candidate.

If your request is successful, you will get a response in JSON format containing the files attached to the dossier. This response is quite extensive with quite a lot of fields (allowing you to pinpoint the document you would like to download) and you will need the field ‘**id_rec**’ to actually download the file.

As you might have seen in the initial request, you are retrieving the first 100 files (limit ‘100’, offset ‘0’). Most dossiers will have (much) less than 100 files. If the dossier however has over 100 document types you can either increase the ‘limit’ in your request or loop through results using a total count (so that if your first response includes a total count of 255, you can do two additional requests (offset ‘100’ & offset ‘200’) to retrieve all results.

To retrieve this total count, you will need to do a POST request using a JSON request to the OWS endpoint https://ows.otys.nl/jservice.php (please note this is the endpoint you use for all your ‘normal’ OWS requests) which looks as follows:

```
{
	"jsonrpc": "2.0",
	"method": "Otys.Services.DossierService.getCount",
	"params": [
		"[sessiontoken]",
		{
			"objectEntityId": [objectEntityId],
			"objectUid": "[objectUid]",
			"restrictClass": [
				"AttachedDocs"
			],
			"limit": 100,
			"offset": 0
		}
	],
	"id": 1
}
```

As you can see, this request is very similar to the previous one, only difference is that the method is ‘getCount’ instead of ‘getList’ (so other fields should be filled in a similar way).

If your request is successful, you will get a response in JSON format containing which looks as follows:

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

Based upon the total count of **[result]** you can loop through the results using the previous ‘getList’-method.

Download file

Now you have retrieved a file ID based upon the '**id_rec**' field of the previously mentioned 'getList'-method; you are able to download the file.

You can do so by sending a simple parametrized HTTP request to the following URL:
https://ows.otys.nl/fileServiceDownload?ouid=**[fileouid]**&class=AttachedDocument

For authentication we have a legacy option to send the OWS session token as URL parameter 'sessionId'; however out of security perspective we strongly advise you to send this OWS session token as in the cookie data of HTTP headers as a cookie named 'PHPSESSID' with as value the OWS session token.

3.2.2 Examples using Postman

Assuming you have Postman installed and are somewhat familiar to it, you can so by taking the following steps.

First let's retrieve the files of a specific dossier:

1. Click on 'New request' (new tab) icon on top;
2. Change method-select (on left side of 'Enter request URL') to 'POST';
3. At 'Enter request URL' fill in 'https://ows.otys.nl/fileService.php';
4. Open tab 'Body' below it;
5. Select radio button 'raw' below it;
6. Paste the JSON of service call 'DossierService.getList' as shown in paragraph 3.2.2 in the text area below it;
7. Change the fields **[sessiontoken]**, **[objectEntityId]**, & **[objectUid]** to the appropriate contents as mentioned in paragraph 3.2.2;
8. Click on button 'Send' on top right;
9. Response will be shown at the bottom. Pick a document, copy the value of the 'uid'-field and save it somewhere (for example in Notepad).

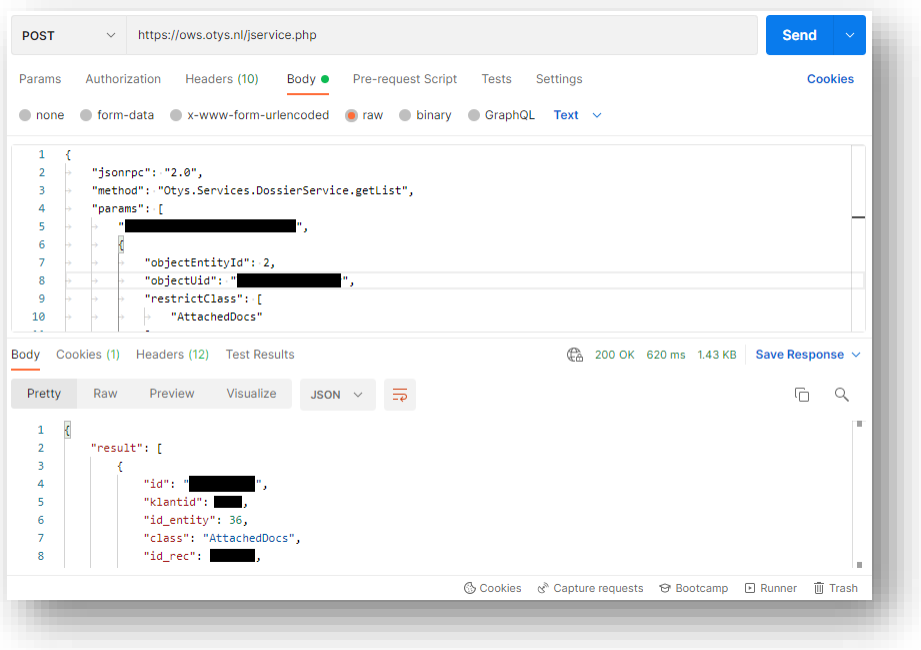

Image 3.2.2.1: Postman request retrieving files from dossier

Now you have the file OUID of the file you wish to retrieve, you can download the file from an internet browser (from example Google Chrome). You can open the following URL in that browser (where you will change **[fileouid]** in the ID of the field 'id_rec' of the previous request):

https://ows.otys.nl/fileServiceDownload?ouid=**[fileouid]**&class=AttachedDocument

Opening this URL 'as is' will return an error, since you did not authenticate yet:

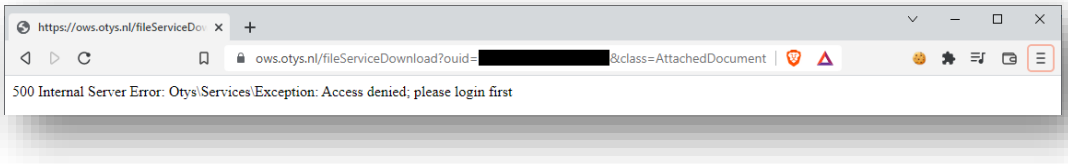

Image 3.2.2.2: Browser file download error without authentication

To be able to download the file; you will need to change the session token which is automatically generated when opening this URL, into the valid OWS session linked to the OWS session that does allow you to download the file. This session token is stored in the cookie named 'PHPSESSID' and can for example easily be adjusted using Chrome plugin EditThisCookie (which can be downloaded from the Google Web Store). Once installed you can change the cooky by taking the following steps:

1. Click on EditThisCookie icon in your browser;
2. Open cookie named 'PHPSESSID';
3. Change 'Value' to your valid OWS session token which has access to the file;
4. Click on 'Submit cookie changes'-icon at bottom of modal.

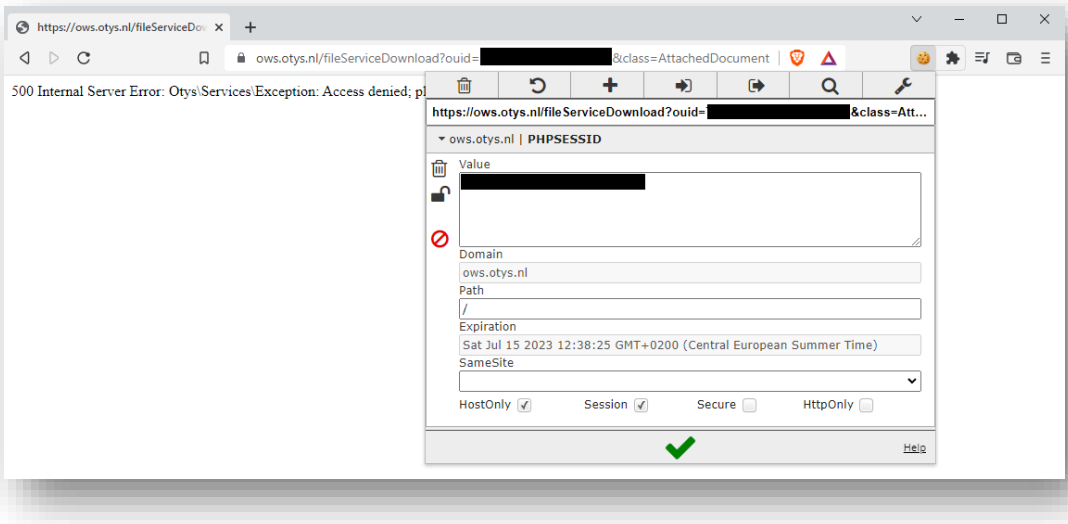

Image 2.2.2.3: Updating session token cookie using EditThisCookie

You can now refresh the page and you will notice the file will be downloaded.

As mentioned before, this example is to show you the process from user interfaces before making code changes. In your final integration you will most probably send the cookie data using the CURLOPT_HTTPHEADER option of cURL or something similar.