

API Justitia.Swiss for Developers

Version: - 36

Projektphase: M4

Erstellungs-Datum: 11.12.2024

Autor: Dominik Auf der Maur, Franz Achermann

Klassifizierung: public

Status: Draft

Table of Contents

1 Testing Environment	5
1.1 Terms of Use	5
2 Tools	6
3 API Use Cases	7
3.1 Submission	7
3.2 Transfer	10
4 API and Security	14
4.1 Create technical Users	14
4.2 Authentication	14
4.3 Authorization	15
4.4 Rate Limiting	15
4.5 Blocking of Requests	15
5 General API Concepts and Guidelines	16
5.1 Missing vs. null vs. empty Properties	16
5.2 Error Handling	16
5.3 File Upload and Download	16
6 API Versioning	19
6.1 Rationale and Goals	19
6.2 Semantic Versioning	19
6.3 Platform API Policies	20
6.4 Compatibility Rules	20
6.5 Technical Implementation of Versioning	21
6.6 References	21
6.7 Implementation Hints	21
7 Frequently asked API-Questions	23
7.1 Update Status	23
7.2 Delete Dossier	23
7.3 Delete Document from Submission	23
7.4 Get all documents	23
7.5 Submissions handling	24

7.6 Handling and Generation of UUIDs	24
7.7 Get Authorities	24
7.8 Versioning	24
8 Breaking Changes Planned for V2	26
8.1 Services	26
8.2 Schemas	26

PROJEKT


Justitia 4.0

This document is intended for (external) developers that need to access the API justitia.swiss (Justitia 4.0). In case of questions, please contact franz.achermann@justitia.swiss.

Version: 11.12.2024 as of Version 1.5.4.

1 Testing Environment

For integration testing of external integrators via API, the TRAIN environment is provided:


<https://platform.train.justitia.swiss/>

As a precondition, the integrator needs to have whitelisted his source IP range with the infrastructure provider, ELCA. In order to be whitelisted, please contact franz.achermann@justitia.swiss

The integrator is expected to register a technical API user for his profile via UI, as explained in [API and Security](#).

1.1 Terms of Use

The following terms of use apply to the TRAIN environment:

-  Do not use productive data on test environments!
- Do not perform load testing on the TRAIN environment. The TRAIN environment is intended for functional tests, not for performance related testing.
- Do not perform penetration testing on TRAIN. You may be blocked or blacklisted, making the platform unavailable to you.

2 Tools

The interactive Swagger UI is available on <https://platform.train.justitia.swiss/swagger-ui/index.html>. The OpenAPI 3.0 spec is here: <https://platform.train.justitia.swiss/v3/api-docs/public>.

Accordingly, the current productive spec is available here: <https://platform.justitia.swiss/v3/api-docs/public>

For the development of API clients, all the well-known **REST client tooling** can be used, depending on the client development environment. Examples of such tools are:

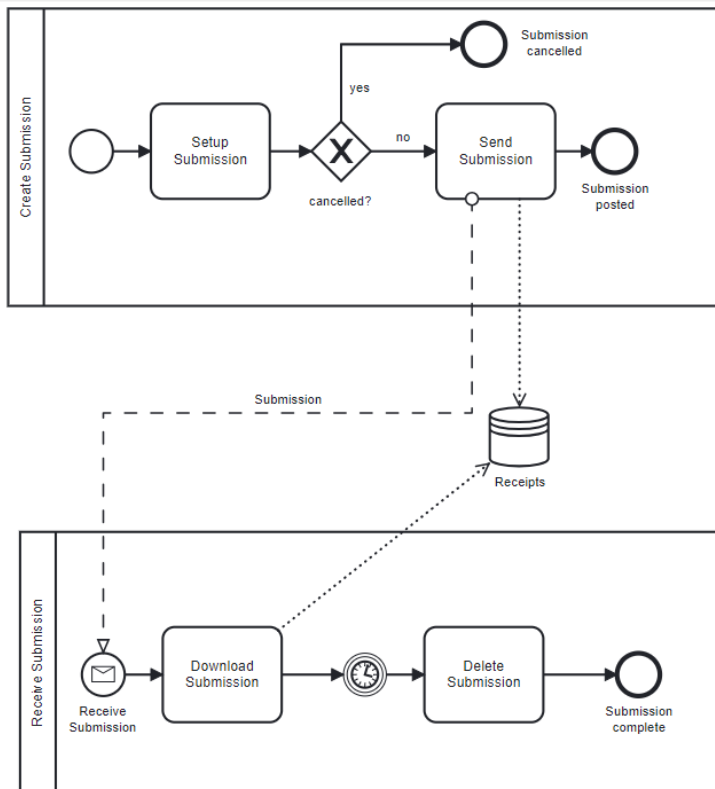
- Java: HTTP Client (built-in with Java 11 and above), Spring WebClient, Apache HttpClient, Square OkHttpClient; in addition, you will require a JSON serializer/deserializer such as Jackson.
- C#: HTTP Client (built-in), RestSharp, Refit
- Python: requests library (built-in), httpx,

If you are relying on client SDK generators such as Swagger Codegen, make sure the generated code complies with the backward compatibility rules defined in [API Versioning](#).

3 API Use Cases

This chapter is for the necessary APIs for the business use cases of the platform.

3.1 Submission



3.1.1 Create Submission (All Participants)

3.1.1.1 Setup Submission

Step
<p>Retrieve all authority profiles as possible recipient of the submission</p> <pre>curl --location 'https://platform.train.justitia.swiss/api/participant/v1/authorities' \ --header 'accept: application/json' \ --header 'Authorization: Bearer *****'</pre> <p>In the response, you find your desired receiver, more precisely, the UUID of it.</p>

Step

Create the new submission.

```
curl --location 'https://platform.train.justitia.swiss/api/document-exchange/v1/submissions' \  
--header 'accept: application/json' \  
--header 'Content-Type: application/json' \  
--header 'Authorization: Bearer *****' \  
--data '{  
  "subject": "myTest",  
  "recipientProfile": "6cfd0c47-8718-4582-b22c-700f81b9272a"  
}'
```

In the response, you find the UUID of your created submission.

NB: you may also need to pass a 'submissionType' or refer to a specific 'dossierId' in the passed data.

Future extension (PF-286): we plan a mechanism such that you can associate a received transmission to a passed submission, if the authority refers to it.

Add a file to the submission. Provide the proper UUID of the submission from the previous step.

```
curl --location 'https://platform.train.justitia.swiss/api/document-exchange/v1/submissions/e9e553f5-6d4e-4d49-b1f3-7783b0017224/documents' \  
--header 'accept: application/json' \  
--header 'Authorization: Bearer *****' \  
--form 'file=@"/tmp/myfile.pdf"'
```

In the response, you find the UUID of the uploaded document. This id is needed to modify the metadata of the document.

Read the current submission:

```
curl --location 'https://platform.train.justitia.swiss/api/document-exchange/v1/submissions/e9e553f5-6d4e-4d49-b1f3-7783b0017224' \  
--header 'accept: application/json' \  
--header 'Authorization: Bearer *****'
```

The response will contain the details of the submission as well as the details of all documents. Note the version of the submission, which is used for the optimistic locking.

Step

Send the current submission. You need to replace the version with the current version of the submission, as retrieved in the previous step.

```
curl --location --request PUT 'https://platform.train.justitia.swiss/api/document-exchange/v1/submissions/e9e553f5-6d4e-4d49-b1f3-7783b0017224/status' \
--header 'accept: application/json' \
--header 'Content-Type: application/json' \
--header 'Authorization: Bearer *****' \
--data '{
  "status": "SUBMITTED",
  "version": 3
}'
```

3.1.2 Receive Submission (only authorities)

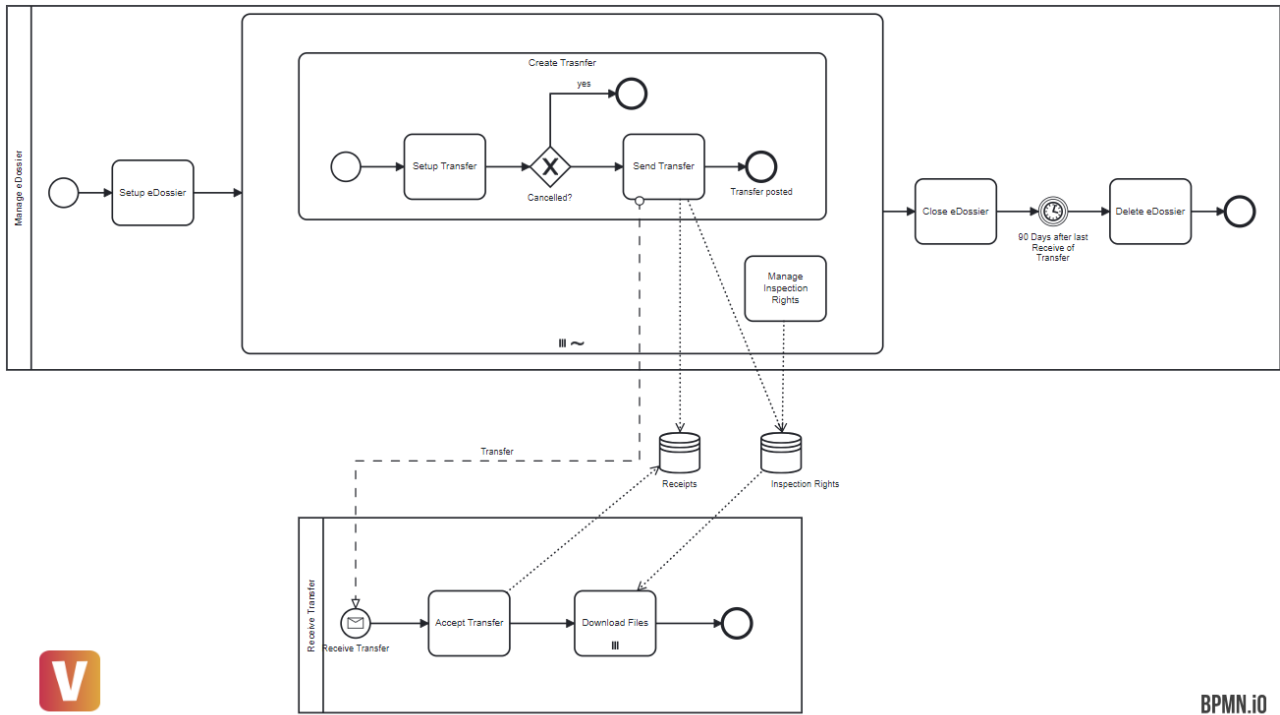
3.1.2.1 Download Submission

Step	API Call
Get all the new transmission (by filtering the inbox on status 'SUBMITTED') Note: the submissions have the same type 'submission'	GET /api/document-exchange/v1/transmissions/inbox
Set the state of the submission to 'RECEIVED' in order to download the contents.	PUT /api/document-exchange/v1/submissions/{submission-id}/status
Get the details of the submission	GET /api/document-exchange/v1/submissions/{submission-id}
Get a file of the submission	GET /api/document-exchange/v1/submissions/{submission-id}/documents/{document-id}
Get all files of the submission	GET /api/document-exchange/v1/submissions/{submission-id}/all-documents

Note: after 90 days, the transmission will be deleted automatically.

On the web-portal, setting the state to 'RECEIVED' is done behind the scenes by the application.

3.2 Transfer



3.2.1 Manage eDossier (only authorities)

3.2.1.1 Setup eDossier

Step	API Call
Create a new dossier	POST /api/document-exchange/v1/dossiers
Retrieve (selected) dossiers	GET /api/document-exchange/v1/dossiers

Note that the profile owning the dossiers is given by the technical user, hence dossiers 'proceeding-number' is unique per profile.

3.2.1.2 Setup Transfer

For a given Dossier (with the dossierId) transfers may be initiated and files uploaded to the dossier.

All Transfer happen in the context of a dossier containing the files:

Step	API Call
Add a file to the dossier:	
<ul style="list-style-type: none"> Upload the file 	POST /api/document-exchange/v1/dossiers/{dossier-id}/documents
<ul style="list-style-type: none"> Provide Metadata of the file (e.g. <code>displayName</code> or your <code>refKey</code> to the document) <p>Note: Metadata of files that are already part of a transfer, cannot be changed.</p>	PUT /api/document-exchange/v1/dossiers/{dossier-id}/documents/{document-id}
Get the details of the dossier (dossier metadata and files metadata)	GET /api/document-exchange/v1/dossiers/{dossier-id}

If you have a dossier, you can create a transfer and add files by referring to the 'documentId' of the dossier (dossierId)

Step	API Call
Lookup the <code>profileId</code> of the recipient	GET /api/participant/v1/profiles
Create a new transfer by providing the <code>profileId</code> of the recipient.	POST /api/document-exchange/v1/transfers
Link files (of the dossier) to the transfer	PATCH /api/document-exchange/v1/transfers/{transfer-id}/document-grant
Get the details of the transfer	GET /api/document-exchange/v1/transfers/{transfer-id}
Cancel (delete) the transfer	DELETE /api/document-exchange/v1/transfers/{transfer-id}

3.2.1.3 Send Transfer

Step	API Call
Send a transfer by updating the status to 'SUBMITTED'	PUT /api/document-exchange/v1/transfers/{transfer-id}/status
Retrieve 'Eingabequittung' after sending the transfer, i.e. by providing status <code>SUBMITTED</code>	GET /api/document-exchange/v1/transfers/{transfer-id}/receipts/{status}

3.2.1.4 Manage the Dossier Permission

A 'Dossier Permission' is the relation between a recipient and a dossier. It holds the validity end date (or undefined) until when the recipient may access any documents of the dossier. The individual documents of the dossier are granted via transfer (i.e. by the endpoint PATCH /api/document-exchange/v1/transfers/{transfer-id}/document-grant).

Step	API Call
Retrieve all active dossier permissions (<code>profileId</code>) of this dossier	GET /api/document-exchange/v1/dossiers/{dossier-id}/permissions
Set the validity period for this recipient.	PATCH /api/document-exchange/v1/dossiers/{dossier-id}/permissions/{dossier-permission-id}
Remove a dossier Permission	DELETE /api/document-exchange/v1/dossiers/{dossier-id}/permissions

3.2.1.5 Close eDossier

Step	API Call
Verify if this dossier may be deleted (i.e. it does not contain any not-yet received transfers)	GET /api/document-exchange/v1/dossiers/{dossier-id}/deletability
Delete this document	DELETE /api/document-exchange/v1/dossiers/{dossier-id}

3.2.2 Receive Transfer (All Participants)

3.2.2.1 Accept Transfer

Step	API Call
Get all the new transmissions (by filtering the inbox on status SUBMITTED) Note: the transfer has the same type TRANSFER	GET /api/document-exchange/v1/transmissions/inbox
Get all the transmissions in my outbox	GET /api/document-exchange/v1/transmissions/outbox
Accept this transfer by passing the status 'RECEIVED'	PUT /api/document-exchange/v1/transfers/{transfer-id}/status

3.2.2.2 Download Files

Step	API Call
Get the details of the transfer	GET /api/document-exchange/v1/transfers/{transfer-id}
Get receipts of the transfer	GET /api/document-exchange/v1/transfers/{transfer-id}/receipts/{status}
Get a file of the dossier the transfer relates to	GET /api/document-exchange/v1/dossiers/{dossier-id}/documents/{document-id}
Get all files of the transfer	GET /api/document-exchange/v1/transfers/{transfer-id}/all-documents

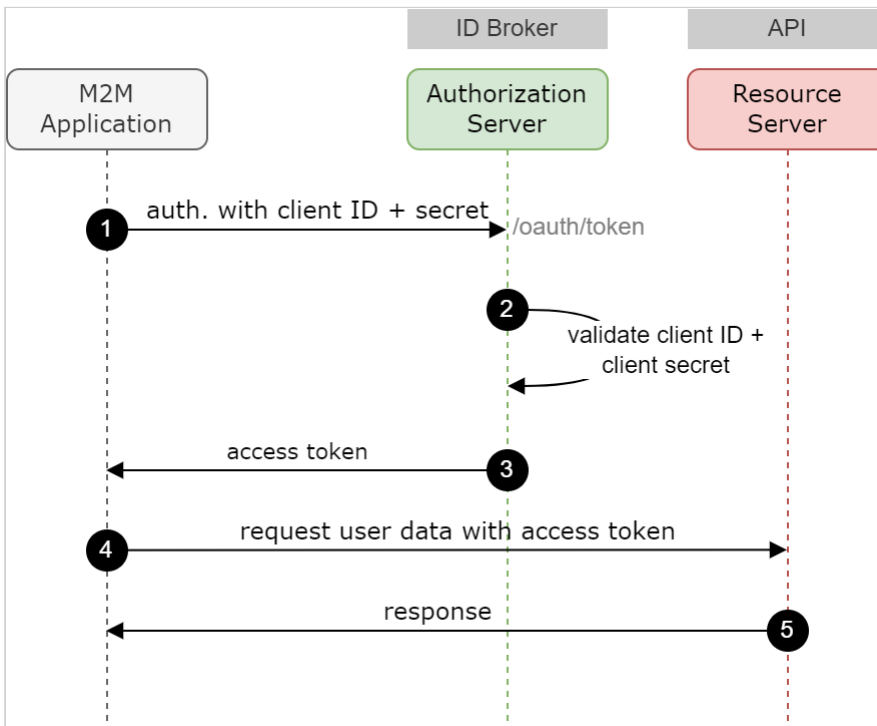
4 API and Security

4.1 Create technical Users

A member of the profile with the Role 'Technical Administrator' can create technical users. Note that only the Administrator of the profile can set give the role 'technical administrator'. For a newly created profile, this role is not available: you have to opt-in explicitly by assigning the role 'technical administrator'.

4.2 Authentication

Each human and technical user that accesses the APIs provided by the "Justitia.swiss" platform must be authenticated. For regular users, the authentication is conducted by the SPA. For technical API users, we provide functionality via UI (accessible to the organisation admin) to register a specific OIDC client on the platform's Keycloak broker using dynamic client registration. Once set up, the API technical user can then use this client to authenticate with Keycloak using the **OIDC Client Credentials** flow.



1. The application authenticates with the Authorization Server using its Client ID and Client Secret (/oauth/token endpoint).
2. The Authorization Server validates the Client ID and Client Secret.
3. The Authorization Server responds with an access token.
4. The application can use the access token to call an API on behalf of itself.
5. The API responds with requested data.

Note: The Client Credentials flow assumes that the integrator is able to securely store the credentials obtained when registering the technical user via UI. Since the Justitia.Swiss platform does not store the credentials, a new API technical user has to be generated in case the credentials are compromised or lost.

The following call gets the access token server, client_id and client_secret.

```
curl --location 'https://platform.train.justitia.swiss/auth/realms/JustitiaPlatform/protocol/openid-connect/token' \  
--header 'Content-Type: application/x-www-form-urlencoded' \  
--data-urlencode 'grant_type=client_credentials' \  
--data-urlencode 'scope=openid profile roles' \  
--data-urlencode 'client_id=****' \  
--data-urlencode 'client_secret=****'
```

After authentication, the API clients must send the **JWT** obtained before as a **bearer token** in each request (stateless) in a standard `Authorization: Bearer <token>` HTTP header.

Note that the access token returned as authentication result has an **expiry**. Using an expired bearer token will yield an HTTP error status 401 "Unauthorized". Therefore, the client is expected to check expiry and when necessary obtain a new access token by another Client Credentials authentication request to the Keycloak broker. The token expiry is stored in the standard JWT `exp` claim.

4.3 Authorization

Authorization of technical API users is **identical to regular UI users**:

- Each technical API user is linked to exactly one profile.
- A technical API user has access to all ordinary business functionality, but not to organisation management related features. These are available to the organisation administrator via UI only.

4.4 Rate Limiting

The following rate-limiting for specific API endpoints will be employed:

- Number of requests (based on leaky bucket algorithm)
- Number of requests to APIs by a given count per time (fixed window algorithm)
- Number of concurrent requests to your services

The specific rates are determined based on load tests on the platform as this gives a good indication on how many requests the platform can handle. This can be further adapted during the operation of the platform.

4.5 Blocking of Requests

In addition to not exposing internal APIs, we will also explicitly block them in the API gateway.

5 General API Concepts and Guidelines

5.1 Missing vs. null vs. empty Properties

- Inbound: The backend does not distinguish between JSON properties with null values and missing properties. Both are regarded as non-existent.
- Outbound: Properties with null values are in general omitted.
- Collections, outbound: Where semantically correct, the backend returns empty collections instead of omitting properties. E.g., a dossier without files will contain an empty file list.

In certain contexts, empty string values and empty collections have a special meaning when used as (inbound) parameters. E.g., PATCH calls uses empty string values and collections to blank out an existing value. Such usages will be pointed out in the API documentation.

5.2 Error Handling

All API services return, in addition to the HTTP Status Code, a JSON error object in case something goes wrong. The different values of the `errorCode` property are described in the OpenAPI documentation - navigate to the `ErrorCode` object in the schema section.

Note that the platform adds an HTTP header field `X-Request-ID` to all inbound requests and returns it with the response. The value of this field should be reported when support tickets are created, as it allows to trace requests in the log and therefore greatly simplifies error analysis.

5.3 File Upload and Download

5.3.1 Filenames and Paths

- Filenames and paths are limited in size (currently 220 character each), and the maximal path + filename size is limited to 260 characters. The rationale is to stay as compatible as possible with different file systems when downloading files or extracting ZIPs.
- Filenames and paths are sanitized, replacing problematic characters such '<' or ':' as by an underscore.
- Path fragments of filenames are stripped. If paths are to be used, they have to be set via the "path" property in the file metadata.
- Backward slashes are replaced by forward slashes in path names.
- Leading and trailing slashes in paths are removed.

5.3.2 Regular File Upload

Small files with a size up to 100 MB can and should be uploaded via a standard MIME Multipart Request containing a single part with the file content. Note that the file upload API requires a proper Content-Disposition header in the part with a filename attribute. This filename will be stored in the file metadata on the platform.

Multipart Upload Request

```
POST /api/document-exchange/v1/dossiers/da7f950f-d641-401f-a428-164b6e6a2ed6 HTTP/1.1
```



```
Content-Type: multipart/form-data; boundary="----=Part-
f53aac0c-66fe-4de2-810c-86bb31107c2d"
Content-Length: 8736428

-----=Part-f53aac0c-66fe-4de2-810c-86bb31107c2d
Content-Disposition: form-data; name="file"; filename="myfile.pdf"
Content-Type: application/pdf

%PDF-... <file data>
-----=Part-f53aac0c-66fe-4de2-810c-86bb31107c2d--
```

5.3.3 Chunked File Upload

Files larger than the 100 MB must be uploaded in chunks. The mechanism is similar to cloud storage APIs such as <https://cloud.google.com/storage/docs/performing-resumable-uploads>. We show an example for a dossier file.

Chunked Upload

```
# 1. Initiate chunked upload
curl "https://platform.train.justitia.swiss/api/document-exchange/v1/dossiers/
5a63cf0e-1915-4905-8498-fa6264a7e223/chunked-documents" \
  -H "Authorization: Bearer ${token}" \
  -H "Content-Type: application/json" -X POST -d '{ "displayName": "myfile.pdf" }'

# Let's say this yields document id=a1b2da02-7020-4b7f-b6da-99d60e8beff0 and
chunkSize=67108864

# 2. Upload chunks like this
curl "https://platform.train.justitia.swiss/api/document-exchange/v1/dossiers/
5a63cf0e-1915-4905-8498-fa6264a7e223/chunked-documents/a1b2da02-7020-4b7f-
b6da-99d60e8beff0" \
  -H "Authorization: Bearer ${token}" \
  -H "Content-Range: bytes 0-67108863/246876120" \
  -F "data=@my-235mb-file.pdf.chunk0

# The response will have isChunked=true and isComplete=false until the last chunk was
uploaded
```

Step 2 is done for each chunk and does not need to be in order. Failed chunk uploads can be re-tried. However, successfully uploaded chunks are never overwritten. Subsequent uploads of already existing chunks are simply ignored by the platform.

Unlike for regular file uploads, the chunks' filenames that may be part of the `Content-Disposition` header are ignored. The only relevant filename is the "displayName" provided in the initialization request. This filename, like any other filename, must be unique in combination with a potential path.

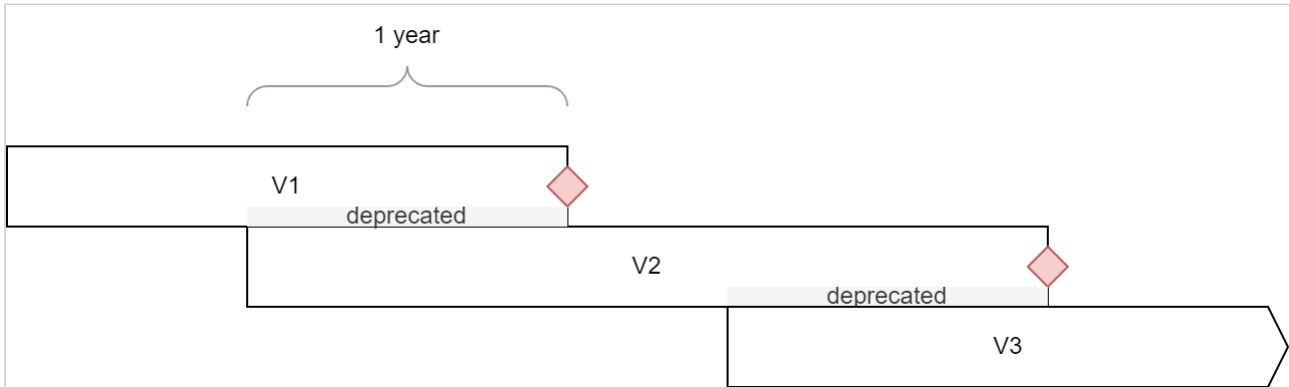
Note that the platform performs strict validation checks:

- The total size specified in the Content-Range header must be correct and must be identical for each chunk.
- The chunks must have the expected size.
- The chunk ranges must be aligned with the chunk size. Also note that the upper bound is (by HTTP header definition) inclusive.

6 API Versioning

6.1 Rationale and Goals

External integrators cannot be forced to do coordinated "big bang" migrations when the platform API changes in a non-backward-compatible way. For this reason, we version the API and support up to two versions of the platform API simultaneously. The following picture shows the typical scenario.



Once a new API version is introduced, the existing one is deprecated but provided for 1 more year. During this period, the integrators are expected to migrate to the new API version.

Note that during the lifetime of an API version, there may be backward compatible changes to the API. For this reason, it is important to implement API clients according to the compatibility rules described below.

6.2 Semantic Versioning

The API is versioned using a `major.minor.patch` version scheme according to Semantic Versioning - see e.g. <https://semver.org/>. This version number is reported in the OpenAPI documentation. The major version number is the one that is also part of the REST path component. This allows to run multiple (incompatible) major versions of the API in parallel. Because minor version updates are deployed without special prior notice, it is important that the API clients respect the compatibility rules - see "Compatibility Rules" below.

Note that the API version and its Semantic Versioning approach is not directly related to the more business oriented Justitia Platform version number. The following scenario is theoretically possible:

Point in time	Platform version	API version	API Path component
27.09.2024	1.4.2	1.7.3	api/.../V1/...
12.12.2024	1.6.3	1.9.12	api/.../V1/...
15.05.2024	1.9.4	2.1.1	api/.../V2/...

6.3 Platform API Policies

Deprecation

Interfaces or parts thereof (operations, data elements, ...) that are deprecated are documented. In OpenAPI, the keyword `deprecated` is used for this purpose. The documentation explains the reasons and points to the "new" (replacement) mechanism or data element.

Decommissioning

A deprecated interface is still supported for 1 year before it is decommissioned, i.e., taken out of operation.

6.4 Compatibility Rules

The **platform client** must be built in a **forward compatible** way since certain changes on the platform side may be activated without special notice. In detail, the following changes must be accepted and ignored by the consumer:

- Added REST services must be ignored
- Additional properties in JSON response objects must be ignored ("tolerant reader" pattern)
- Depending on case: unknown enum values in response should be ignored ("tolerant reader" pattern)
Remark: Whether or not this type of change can be interpreted as backward compatible depends on the concrete case and will be mentioned in the documentation.
- Additional enum values used as input parameters must be acceptable
Remark: However, re-interpretation of existing enum values is not backward compatible.
- Added optional query parameters must be ignored
- Removed input parameters or properties on input objects must not break the client

Code Generators

Note that client SDK generators such as Swagger Codegen in general do a bad job at generating forward compatible libraries. E.g., Swagger Codegen for Java does not even ignore additional response properties.

In contrast, the client is not expected to accept changes such as:

- Removed mandatory property in response
- Additional mandatory input parameter
- Property renaming

Such types of changes will be introduced in a new major API version only.

Semantics and Compatibility

The above rules are simple to understand and guarantee compatibility on a syntactic level. In reality, the more difficult questions about compatibility have to be asked on higher semantic levels. We try to convey as many of those business rules as possible in the description. When deciding whether or not a platform API change is breaking and should lead to a major version change, we do not focus on syntax alone. Instead, semantic changes (e.g. using a field in a different way) may also lead to a major API version update.

6.5 Technical Implementation of Versioning

We implement **URI Path Versioning**: REST resources are versioned by introducing a major version number in the URI path, directly after the domain name:

<https://hostname/api/domain/Vn/...>

A concrete example would be:

<https://justita-swiss.ch/api/participant/V1/profiles/2b0937d2-f165-450c-b947-f151f111c953>

The version is updated if and only if a change is breaking.

The **scope** of one version is a domain, so there is a separate version for the "participant" and "document-exchange" parts of the API.

6.6 References

The approach presented here is one of the best practices, followed by large players such as Salesforce. There are many resources on the Web, some of them are:

- <https://blog.hubspot.com/website/api-versioning>
- <https://restfulapi.net/versioning/>
- <https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design#uri-versioning>

6.7 Implementation Hints

Ignore added REST services

No special measures necessary. With or without generated client SDKs, the additional operations will just remain unused.

Ignore additional properties in response

Usually, the object mapper can be configured to ignore unknown properties on de-serialization.

Ignore unknown enum values in response or input parameter objects

Just ignoring unknown enum values is the wrong choice in many situations. If the consumer expects to handle all enum cases exhaustively, ignoring new cases would lead to undetected failures.

Solution A: Map the field to a string instead of an enum on the client side, even if the field is declared as an enum type in OpenAPI. In the client logic, do the appropriate validation and throw an error if unknown values are not acceptable.

Solution B: Map the field to an Enum type, but parse the JSON string in such a way that all unknown values are mapped to an Unknown value defined in the Enum. In the client logic, handle Unknown appropriately.

Note: Standard code generators such as Swagger Codegen generate Enums without further measures and are therefore not tolerant!

Don't fail due to additional enum values in input

No special measures necessary. With or without generated client SDKs, the additional enum values will just remain unused.

Don't fail due to additional (optional) input parameters

No special measures necessary. With or without generated client SDKs, the additional parameters will just remain unused.

Don't fail due to removed input parameters or input properties

We assume that removing the input parameter does not affect functionality. E.g., we will not remove a filter parameter such as limit and offset used for paging and then just return all data.

The client may emit excess parameters, but since the platform also respects the "tolerant reader" pattern, this data is just ignored.

7 Frequently asked API-Questions

- [Update Status](#)
- [Delete Dossier](#)
- [Delete Document from Submission](#)
- [Get all documents](#)
- [Submissions handling](#)
- [Handling and Generation of UUIDs](#)
- [Get Authorities](#)
- [Versioning](#)

7.1 Update Status

Q: Why don't you use patch method to update status ?

- You create a specific status resource for managing it.
- The status is one property of the resource submission or Transfer
- Why don't you use the patch verb to update the status like other patch method on others resources

A: It's a tradeoff of readability and understandability on the swagger.

7.2 Delete Dossier

Q: Why don't you use the OPTION verb to know if the resource can be deleted ?

- On the dossier resource you define a delectability sub resource to know if the dossier can be deleted
- The OPTION verb can be used to get info about action that can be allowed on a resource

A: It's a tradeoff of readability and understandability on the swagger.

Q: Could we understand that the deletion of one dossier is only scheduled for a latter batch deletion ?

A: We need to keep the document (according to E-BEKJ) for 90 days, hence 'closing' a dossier makes it immutable and the physical deletion happens in a batch..

7.3 Delete Document from Submission

Q: Why is there no 'delete' endpoint to delete a single file from a dossier:

A: we expect the client system to send the right files. More precisely, we assume the business use case is only started by the client, when all the files are known to the client system.

7.4 Get all documents

Q: the method used to retrieve all documents on a transfer or submission should use an additional resource:

- Why don't you use the standard url (/api/document-exchange/v1/submissions/{submission-id}/documents) with a specific accept header
- This allows client to request resource in his preferred format. It can put "accept : application/json" http header to get the json representation of the list of documents or "accexp: application/zip" http header to get the zip representation of all documents"

A: it might be the case that we put other information (like structure infos) into a zip, hence to encode 'what the clients' expects seems more clear.

7.5 Submissions handling

Q: There is no similar endpoint to '/api/document-exchange/v1/transmissions/inbox' to retrieve all sent transmissions, e.g. the outbox.

A: This is available with Version 0.5.16

7.6 Handling and Generation of UUIDs

Q: All resource identifiers are uuids. Why do you use the POST method to create resources (like submission or document in submission ...). The post method is not idempotent and in case of error during creation, if we have to send another request, we potentially will create duplicated resources. Why don't you use the PUT verb directly to create resources and let the client set the unique identifier. You can respond with the 201 status for resource creation or 200 for resource modification. In case of network pb, if we have to resend the request we don't create new resources.

A: for security reasons we want to 'invent' the uuid on the platform side to make it harder for adversaries to guess uuids. The api allows the 'refKey' on documents to keep your own technical identifier. All the other uuids that the model uses, do not need to be persisted for a long time on the client side. The only key introduced by the platform is the delivery address.

7.7 Get Authorities

Q: Why is there no pagination on authorities services? (/api/participant/v1/authorities

A: Contrary to the inbox, the list of authorities is (for the time being) small enough.

7.8 Versioning

Q: Some resources have a version property. How is this version managed ?

A: The versioning is used for optimistic locking wrt. the Web-interface. Whenever you update a (patch) a resource, you need to pass the proper version, in order to *show* that you know the current state of the resource. For instance, when you send a submission, it must be of the proper 'version'.

Get_Submission (Id = e9e553f5-6d4e-4d49-b1f3-7783b0017224) returns:

```
{
  "id": "7b239523-fe70-47bb-965a-5a02eddd7fbd",
  "version": 1,
  "subject": "Test mit Thomas",
  "creatorProfileId": "479c9954-bf08-410c-aae8-7cf6988a7d1a",
  "recipientProfileId": "6cfd0c47-8718-4582-b22c-700f81b9272a",
  "status": "DRAFT",
```



```
"createdAt": "2023-11-30T16:27:35.576Z",  
"documents": [ ... ]}
```

Note that the version is 1. When we try to send this submission (using PUT submissions/{{submission_id}/status), we need to pass this version number. Otherwise we would get an 409 Conflict error.

8 Breaking Changes Planned for V2

8.1 Services

8.1.1 document-exchange/dossier

Operation	Path	Change	Explanation
PUT	/api/document-exchange/v1/dossiers/{dossier-id}/documents/{document-id}	Service removed	For API consistency reasons. Use PATCH on same path instead.

8.1.2 document-exchange/submission

Operation	Path	Change	Explanation
PUT	/api/document-exchange/v1/submissions/{submission-id}/documents/{document-id}	Service removed	For API consistency reasons. Use PATCH on same path instead.

8.2 Schemas

Type	Property	Change	Explanation
SubmissionDto	createdAt	Declared optional (now: mandatory)	This property will be hidden from the submission recipient. We will remove the current workaround of returning a dummy timestamp.
TransmissionDto	createdAt	Declared optional (now: mandatory)	This property will be hidden from the submission recipient in case of a Submission. We will remove the current workaround of returning a dummy timestamp.