SERVICEWARE

SABIO 5 REST API Documentation

Last Published: 2021-04-22 | Version: 2.0.11

Table of Contents

Overview

This chapter explains how to read this documentation, documents the REST service in general and introduces basic concepts. For example, how to authenticate against the REST service.

Code Formatting Conventions

All code snippets in this documentation underlie the following formatting conventions:

Concept	Examples	Description
Source code fragment	`filter`, `/tree`, `false`	All fragments of source code - a variable, a REST resource path, literals … - are written in mono-spaced font.
Standard JavaScript type	`<NUMBER>`, `<STRING>`	All standard JavaScript types are written in capitals, encapsulated with `<` and `>`
Resource Entity	`<TreeResource>`, `<TextResource>`	All REST resource entities are written in "camel case", encapsulated with `<` and `>`
Placeholder	`__USER_NAME__`, `__UUID__`	All placeholders are written in capitals, beginning and ending with two underscores. Words are separated by a single underscore.

REST Service Entry Point

The entry point for all services is accessible from one, globally defined URL. To prevent cross-site errors in web browsers, the service must reside in the same domain as the calling client. That is, each realm uses its own service URL.

If not called by web browsers, the service may be reachable through a different domain. This depends on the security policy of the client device.

The generic form of the REST service entry point looks like:

```
__PROTOCOL__://__SUBDOMAIN__.__HOSTNAME__.__TLD__/__APPLICATION_CONTEXT_PATH__/__SERVICE_BASE_PATH___/__SERVICE_NAME__
```

where __APPLICATION_CONTEXT_PATH__ defaults to sabio depending on installation and __SERVICE_BASE_PATH___ defaults to services.

Whenever a service is defined, the REST service entry point is omitted for sakes of readability. Given your REST service entry point is https://mycompany.sabio.de/sabio/services and you want to request a <TreeResource> via the /tree service - as stated by the REST API reference - the effective URL would be:

```
https://mycompany.sabio.de/sabio/services/tree
```

Calling the REST API

In general, the provided services allow for

- requesting lists of all available resources or
- requesting single resources by passing a UUID in the service URL.

The call to a "write operation" (POST, PUT) always returns the written resource. Note that creating a resource via POST never requires a UUID.

Querying the REST API follows the general conventions for RESTful services, where each CRUD operation - create, read, update and delete - is mapped to its corresponding HTTP method:

- Requesting a single resource identified by the provided UUID.
  - @GET
  - SERVICE_NAME__/__UUID__
- Creating a new service resource. The resource fields are transmitted in the request body.
  - @POST
  - /__SERVICE_NAME__
- Deleting the resource identified by the provided UUID.
  - @DELETE
  - /__SERVICE_NAME__/__UUID__
- Modifying a single resource identified by the provided UUID. The resource fields are transmitted in the request body.
  - @PUT
  - /__SERVICE_NAME__/__UUID__

Authentication

See section *“User Authentication”* in *“Standard Services”*.

Response Format

The REST API returns JSON only.

For writing calls (PUT, POST), the Content-Type: application/json header field has to be set. In some cases the response body contains a JSON encoded string, but with a content header set to text/plain (e.g. after uploading files). However, such exceptions are documented in the REST API reference.

Generic Response Object

With exceptions documented in the REST API reference, all services implement a generic response object that implements the following structure:

```
{
  "data": {
    "result": <NULL>|<Resource>|<Resource>[]
  },
  "status": <Status>
}
```

- The data property makes the response accessible.
  - Its result property holds the actual response payload, containing either
    1. a single <Resource> entity,
    2. an array of <Resource> entities or
    3. <NULL>, if the request failed.
  - Optionally, the data property may contain properties for convenient result processing, e.g. providing the total number of results for a list of <Resource>s in a total property field
  - See the API Reference for detailed documentation about the concrete returned <Resource> entities of each service
- The status property contains information, whether the request could be successfully handled by the server.

See section *“Standard Resources and Entities“* for more details.

Request/Response Field Values

All fields in the request/response body may only contain the following JavaScript types:

- <STRING>
- <NUMBER>
- (<INTEGER> - used for sakes of clarity to express API intent; handled as <NUMBER> internally)

- (<FLOAT> - used for sakes of clarity to express API intent; handled as <NUMBER> internally)
- <BOOLEAN>
- <ARRAY>
- <OBJECT>
- <NULL>
- <UNDEFINED>

A field that is declared as UUID is of type <STRING> and has a length of 32 characters. An UUID uniquely identifies a single resource *across all services*, that is, all resources share a “global ID namespace”.

A <DATE> is a <STRING>-field that must match one of the two patterns (where the letters have the meaning specified in SimpleDateFormat)

- EEE MMM dd yyyy HH:mm:ss 'GMT'Z - For example "Sat Mar 13 2010 23:29:05 GMT+0200". This is also the format that the REST API emits (e.g. in the created-fields, in case a new /text resource entity is created).
- EEE, dd MMM yyyy HH:mm:ss Z (compliant with RFC 1123) - For example "Wed, 02 Oct 1991 22:59:00 GMT" or Tue, 20 Feb 18 15:01:52 -0100 using a numerical offset to UTC.

Further data types are defined as resources. Function calls or function definitions are not allowed.

Error Handling

Any errors at service level will be mapped to the corresponding HTTP status, for example, returning 403 FORBIDDEN if someone makes unauthorized service requests.

See *“Standard Resources and Entities”* for details.

Caching

The validity of the data provided by the service refers to the point of delivery. To prevent client-side caching - especially for web browsers! - the web server has to be configured sending these HTTP header fields:

```
Cache-Control: no-cache, no-store, must-revalidate, max-age=0

Pragma: no-cache

Expires: Thu, 01 Jan 1970 00:00:00 GMT

ROBOTS: NOARCHIVE
```

Versioning

At this point the API doesn’t support versioning.

Standard Resources and Entities

This chapter briefly describes common resources/entities and their purpose. Consult the REST API reference for detailed information and/or valid field values.

Status

Each response contains a <Status>that determines, whether a request succeeded or failed.

Field	Type	Description
httpCode	Integer	The HTTP status code
code	String	Beta: In the future, this field will contain a technical, four-character hex-status-code (error codes will start with an f when converted to a HEX value). Currently, this field contains the HTTP code or an empty string.
text	String	A description of what has happened.
success	String	Determines, if the service call was successful. In general, an HTTP status between 200 and 399 is considered to be a success.

Example of a successful response:

```
{
  "status": {
    "success": true,
    "code": 0,
    "httpCode": 200,
    "text": "Request successful submitted"
  }
}
```

Example of a failed response:

```
{
  "status": {
    "success": false,
    "code": 62465,
    "httpCode": 401,
    "text": "Authorization required"
  }
}
```

LightGroupResource

A LightGroupResource is a resource with reduced properties of a group and used in an another resource, e.g. in resource of /tree service.

Example:

Name	Type	Read-Only	Required for POST	Description
id	UUID	yes	no	The resources UUID.
name	String	no	yes	The name of the user group.

LightUserResource

A LightUserResource is a resource with reduced properties of an user and used in an another resource, e.g. in resource of /tree service.

Example:

Name	Type	Read-Only	Required for POST	Description
id	UUID	yes	no	The resources UUID.
firstname	String	no	yes	The first name of the user.
lastname	String	no	yes	The lst name of the user.

LightViewResource

A LightViewResource is a resource with reduced properties of a view and used in an another resource, e.g. in resource of /text service.

Example:

Name	Type	Read-Only	Required for POST	Description
id	UUID	yes	no	The resources UUID.
title	String	no	yes	The title of the document.

LightTreeNodeResource

A LightTreeNodeResource is a resource with reduced properties of a tree node and used in an another resource, e.g. in resource of /text service.

Example:

Name	Type	Read-Only	Required for POST	Description
id	UUID	yes	no	The resources UUID.
title	String	no	yes	The title of the document.

LightDocumentResource

A LightDocumentResource is a resource with reduced properties of a document and used in an another resource, e.g. in resource of /text service.

Example:

Name	Type	Read-Only	Required for POST	Description
id	UUID	yes	no	The resources UUID.
title	String	no	yes	The title of the document.

Authentication with OpenID Connect (OIDC)

Preface

OpenID Connect (and its base OAuth2) is a common industry standard for doing authentication as well as for identity management. One of it's core characteristics is a dedicated service ("Auth Server") that deals with several ways of (user) logins. As a result of a login, an access token is issued. Such token can be used for accessing REST API of application ("Resource Server") where this token has been issued for.

Authentication with OpenID Connect is supported by SABIO with Keycloak as an Auth Server implementation. While previously described ways for authentication (such as classic username / password, api key authentication , …) are still supported as well, these will be removed on long term and entirely replaced with OpenID Connect authentication.

A lot of public documentations (such as this) exist for how to authenticate with OpenID Connect and its different flows. For this reason, this documentation does not provide a deeper dive into OIDC's core concept and particularities but aims to provides examples for how to do authentication for SABIO Knowledge only.

General Authentication Steps

This sections describes general authentication steps that are valid across all OIDC authentication flows. Specific steps for certain flows are described separately below separately in sections "*Authentication flow for …*".

Preparation: Realm Setup (to be carried out by SABIO)

In order to be able to use OIDC authentications at all, some preparations and configurations need to be done. Please contact SABIO support in order to initiate this setup. Anyway, this documentation provides some hints (for *internal* usage) for what to do exactly

1. Keycloak integration needs to be enabled at all for particular SABIO Knowledge realm. This requires to setup a Keycloak realm as well and to migrate existing SABIO users into this Keycloak realm.
2. An additional client (e.g. mycompany_myapplication) needs to be set up and configured for newly created Keycloak realm. This client represents customer's application/client that is going to authenticate via OIDC for SABIO Knowledge. This client's setup depends on application's specific characteristics (e.g. whether it's an UI or a service) and especially on which OIDC flow (e.g. implicit flow, authorization code flow, …) is going to be used. It's important to carefully choose the right flow and configure the client properly (e.g. redirect_uri) to make authentication flow as secure as possible.

Step 1: Fetch OIDC connection parameters

First of all, certain OIDC URLs need to be fetched from SABIO Knowledge (acting as a registry) via REST. These URLs denote standard OIDC endpoints such as authorization_endpoint or token endpoint.

Example for how to fetch URLs

```
<request>

GET https://mycompany.sabio.de/sabio/services/_client


<response>

HTTP/1.1 200 OK

Content-Type: application/json


{
  "data": {
    "authentication": {


      "openidConnect": {
          "authorization_endpoint": "https://auth.sabio.de/auth/realms/mycompany/protocol/openid-connec
t/auth",
          "token_endpoint": "https://auth.sabio.de/auth/realms/mycompany/protocol/openid-connect/token
",
          "end_session_endpoint": "https://auth.sabio.de/auth/realms/mycompany/protocol/openid-connect/
logout"
      }
    }
  },
  "status": {
    "httpStatus": 200,
    ...
  },
  ...
}
```

Step 2: Fetch Access Token

This is about running actual authentication flow and acquire an access_token. Specific steps for actual flow are described below in sections "*Authentication flow for … -> Fetch Access Token*".

Step 3: Access REST API

After having acquired an access_token, this can be used for accessing SABIO Knowledge's REST API by sending it with every request via header, e.g. like

```
<request>

GET https://mycompany.sabio.de/sabio/services/user/profile

Authorization: Bearer <access_token_here>



<response>

HTTP/1.1 200 OK

Content-Type: application/json



...
```

Step 4: Refresh Access Token

Per design, an access_token has a limited time-to-life (e.g. several minutes) only. Once its TTL has been reached, the token can't be used anymore and would result in an 401 error on SABIO Knowledge side. Thus, an accessing application ("client") needs to take care about refreshing this token before expiration.

A token's expiration date can be found out by inspecting token itself. Every token is encoded as JSON Web Token (JWT) and is internally structured like

```
{

  "jti": "8f404ea0-9070-402b-b89a-5857f290af23",

  "exp": 1554898723,

  "nbf": 0,

  "iat": 1554897823,

  "iss": "https://auth.sabio.de/auth/realms/mycompany",

  "aud": "mycompany_myapplication",

  "sub": "0ad00435-8af8-4e9b-a9ff-b84fb1aa5091",

  "typ": "Bearer",

  "azp": "mycompany_myapplication",

  "auth_time": 1554897823,

  ...

}
```

For instance, the field exp holds expiration date as “seconds since 1.1.1970”. Example value 1554898723 might be translated into Wednesday, April 10, 2019 12:18:43 PM.

The way for keeping a token fresh depends on the actually used specific flow and is therefore described in sections “*Authentication flow for … -> Refresh Access Token*” below.

I. Authentication flow for Single Page Apps (SPA)

For single page web apps it is suggested to use OIDC’s Implicit Grant flow for authentication. While this flow’s steps are briefly described below, it is suggested to use Keycloak’s Javascript adapter rather than implementing these steps ‘by hand’.

Keycloak Client Setup (to be carried out by SABIO)

These settings need to be configured by SABIO when setting up a client in Keycloak

- *Client ID*: Arbitrary ID for denoting this client. Example: mycompany_myapplication
- *Implicit Flow Enabled*: yes
- *Public Client*: yes
- *Redirect Uri*: URI that points to app’s entry point, e.g. https://mycompany.com/my_spa

Fetch Access Token

Authentication is initiated by redirecting SPA to an URL where user is asked for his credentials. This URL is composed like

```
https://auth.sabio.de/auth/realms/mycompany/protocol/openid-connect/auth?
                    response_type=token
                    &client_id=mycompany_myapplication
                    &redirect_uri=https://mycompany.com/my_spa
```

with

- <base_url> is taken from connection parameter authorization_endpoint (see above)
- <client_id> is ID from client/application that has been initially set up in Keycloak (see above)
- <redirect_uri> Registered redirect uri (see above)

Redirecting to this URL will eventually show a login screen where user may submit his credentials such as *login name* and *password*. After that, one or more redirects will take place and above’s redirect_uri is finally reached. This redirect URI is enriched with a access token as URI fragment. This token needs to be extracted from URI then. Example:

```
...
<response>
HTTP/1.1 302 Moved Temporarily
Location: https://mycompany.com/my_spa#access_token=(...)TRmOTMtYmQ2MS04YzQ3MTY3YjIzZ(...)
```

In case of an error, redirect uri would contain a parameter error like https://mycompany.com/my_spa#error=some-error-code instead.

Refresh Access Token

Implicit grant flow needs to be re-run once token has expired because refresh_tokens are issued here for security reasons. As it wouldn't be a good option to force user to re-enter his credentials again and again, this should be done as a silent authentication where an additional parameter prompt=none can be sent like

```
<request>

https://auth.sabio.de/auth/realms/mycompany/protocol/openid-connect/auth?

                    response_type=token

                    &client_id=mycompany_myapplication

                    &redirect_uri=https://mycompany.com/my_spa

                    &prompt=none



...

<response>

HTTP/1.1 302 Moved Temporarily

Location: https://mycompany.com/my_spa#access_token=(...)TRmOTMtYmQ2MS04YzQ3MTY3YjIzZ(...)
```

This forces auth server to not show any UI but use an existing session (e.g. represented by implicitly sent browser cookie) so that no manual login is necessary but an access token can be issued immediately. If no active active session exists, then this flow is answered with an error like https://mycompany.com/my_spa#error=some-error-code.

This silent flow should be executed in a "hidden way" (e.g. using a hidden iframe). Also, the client should take care that session at auth server does not expire, e.g. by running this flow frequently and keeping this session active. Keycloak's Javascript adapter comes with a out-the-box implementation for this.

II. Authentication flow for Mobile Apps

For (native) mobile apps it is suggested to use OIDC's Authorization Code with PKCE flow for authentication. This flow requires several sub-steps that include user interaction as well. While these steps are briefly described below, it is suggested to use an OIDC aware SDK such as AppAuth for iOS and for Android rather than implementing these steps 'by hand'.

Keycloak Client Setup (to be carried out by SABIO)

These settings need to be configured by SABIO when setting up a client in Keycloak

- *Client ID*: Arbitrary ID for denoting this client. Example: mycompany_myapplication
- *Standard Flow Enabled*: yes
- *Public Client*: yes
- *Redirect Uri*: Redirect URI to be defined by customer. While this callback URI may be arbitrary, it is suggested for Mobile Apps to use an URI with custom URI scheme such as myapp123://oidc_callback.

Fetch Access Token

Step A: Generate Code Verifier and Code Challenge

This is about randomly generating a (so called) code verifier and hashed code challenge out of it. This needs to be done on device like

```
String codeVerifier = encodeBase64url(generateRandom64Bytes()); // e.g. eXvTUmTrVunfPmr-0UaIvTjSHJJ
9O9ZZqfWiBOKs3QD-1oddxfRWutkQjxqFbmoxNYIAZyNr91Y_k9DiwW6w_Q

String codeChallenge = encodeBase64url(hashSha256(codeVerifier.getBytes())); // e.g. fFC-SQZc26fo9hIJ6ji
HkeOBcwZC6ADbFBeLAXP8B5M
```

Step B: Initiate Authentication with UI

Authentication needs to be initiated by opening a certain URL in mobile device's web browser UI (such as a web view). This URL is composed like

```
https://auth.sabio.de/auth/realms/mycompany/protocol/openid-connect/auth?

                    response_type=code

                    &client_id=mycompany_myapplication

                    &code_challenge=fFC-SQZc26fo9hIJ6jiHkeOBcwZC6ADbFBeLAXP8B5M

                    &code_challenge_method=S256

                    &redirect_uri=myapp123://oidc_callback
```

with

- <base_url> is taken from connection parameter authorization_endpoint (see above)
- <client_id> is ID from client/application that has been initially set up in Keycloak (see above)
- <code_challenge> is generated code challenge (see above).
- <redirect_uri> is an (arbitrary) callback URI. For mobile apps, it is suggested to use an URI with custom URI scheme such as myapp123://oidc_callback. Note: That this uri is also required to be registered in Keycloak for this client/application!

Opening this URL in a web view will eventually show a login screen where user may submit his credentials such as *login name* and *password*. After that, one or more redirects will take place and above's redirect_uri is finally reached. This redirect URI is enriched with a query parameter code that holds the so called authorization code. Example:

```
...

<response>

HTTP/1.1 302 Moved Temporarily

Location: myapp123://oidc_callback?code=CyvsiQ4fr85oSFpx37FVYFNMQ
```

In case of an error, redirect uri would contain a parameter error like myapp123://oidc_callback?error=some-error-code instead.

Step C: Exchange Authorization Code into Access Token

The received authorization code needs to be exchanged into final access_token immediately. This is done by POSTing it directly (e.g. not via UI / web view) to token_endpoint (see above). Example:

```
<request>
POST https://auth.sabio.de/auth/realms/mycompany/protocol/openid-connect/token
Content-Type: application/x-www-form-urlencoded


grant_type=authorization_code
&client_id=mycompany_myapplication
&redirect_uri=myapp123://oidc_callback
&code=CyvsiQ4fr85oSFpx37FVYFNMQ
&code_verifier=eXvTUmTrVunfPmr-0UaIvTjSHJJ9O9ZZqfWiBOKs3QD-1oddxfRWutkQjxqFbmoxNYIAZyNr91
Y_k9DiwW6w_Q


<response>
HTTP/1.1 200 OK
Content-Type: application/json


{
  "access_token": "(...)TRmOTMtYmQ2MS04YzQ3MTY3YjIzZ(...)",
  "token_type": "bearer",
  "expires_in": 300,
  "refresh_token": "(...)YTktOWYwNi1lODQ1NTk1MjQ5Y(...)",
  "refresh_expires_in": 1800,
  ...
}
```

with

- <base_url> is taken from connection parameter token_endpoint (see above)
- <client_id> (see above)
- <redirect_uri> needs to be exactly same than redirect_uri that is used by above described authorization_endpoint call
- <code> authorization code that is extracted from URL in previous step
- <code_verifier> is generated code verifier (see above).

Keep in mind that parameters have to be encoded as x-www-form-urlencoded and sent via request body according to HTTP specification.

Resulting access token can be extracted from JSON body and used for subsequent REST calls then.

Refresh Access Token

Besides the short-lived access_token, this flow exposes a longer lived refresh_token as well. This can be used for refreshing an expired access token without running re-running above's flow where user has to re-enter his credentials.

```
<request>
POST https://auth.sabio.de/auth/realms/mycompany/protocol/openid-connect/token
Content-Type: application/x-www-form-urlencoded

grant_type=refresh_token
&client_id=mycompany_myapplication
&redirect_uri=<original_redirect_uri_here>
&refresh_token=<refresh_token_here>

<response>
HTTP/1.1 200 OK
Content-Type: application/json

{
  "access_token": "(...)TRmOTMtYmQ2MS04YzQ3MTY3YjIzZ(...)",
  "token_type": "bearer",
  "expires_in": 3600,
  ...
}
```

Note that a refresh_token needs to be kept secret under all circumstances and must not be used or exposed in a shared environment such as a web browser. Also note, that a refresh token has a limited lifetime as well so that it is required to run original flow from time to time.

III. Authentication flow for Backend Services

Above described authentication steps for *Mobile Apps* or *Single Page Apps* are UI based, e.g. real persons have to enter their credentials via a UI. Proposed authentication flows are designed to take special care that credentials are not leaked and accounts are not compromised.

In opposite, backend services can be assumed to run a secured environment where credentials can stored securely. Thus, password grant with additional client_secret protection shall be used here.

Keycloak Client Setup (to be carried out by SABIO)

These settings need to be configured by SABIO when setting up a client in Keycloak

- *Client ID*: Arbitrary ID for denoting this client. Example: mycompany_myapplication
- *Access Type*: confidential
- *Direct Access Grants Enabled*: yes
- Credentials
    - *Client Authenticator*: Client Id and Secret
    - *Secret*: Any unguessable secret. Example: my-client-secret123

In addition, a service user incl. a password needs to be set up in *Users* section.

- *Username*: A login name, e.g. my-service-user
- *Credentials -> Password*: An unguessable password, e.g. my-service-user-password456

Fetch Access Token

Fetching an access_token can be done by a single REST call like

```
<request>
POST https://auth.sabio.de/auth/realms/mycompany/protocol/openid-connect/token
Content-Type: application/x-www-form-urlencoded

grant_type=password
&client_id=mycompany_myapplication
&client_secret=my-client-secret123
&username=my-service-user
&password=my-service-user-password456

<response>
HTTP/1.1 200 OK
Content-Type: application/json

{
  "access_token": "(...)TRmOTMtYmQ2MS04YzQ3MTY3YjIzZ(...)",
  "token_type": "bearer",
  "expires_in": 300,
  "refresh_token": "(...)YTktOWYwNi1lODQ1NTk1MjQ5Y(...)",
  "refresh_expires_in": 1800,
  ...
}
```

with

- <base_url> is taken from connection parameter token_endpoint (see above)
- <client_id> (see above)
- <client_secret> A secret that is bound to client_id
- <username> Name of service user
- <password> Service user’s password

Refresh Access Token

Although this flow exposes a refresh_token, there is no advantage in making use of this. Instead, password grant flow can be re-run once token has expired.

Standard Services

This section briefly describes “general purpose services”. For detailed documentation refer to the REST API reference, e.g. valid values for a certain field.

User Authentication

General Authentication Mechanism

Since the REST service is stateless, the user has to be authenticated with each request. This is achieved by sending an authentication token in the HTTP header field sabio-auth-token. The token is generated by the server and then sent to the client. To authenticate, the client simply sends the token in addition to the the actual request payload for each request. So, the client doesn’t have to do any computation with the token sent by the server.

To learn more about the /authentication service - for example supported authentication methods - consult the REST API reference.

Requesting an Authentication Token (POST method)

Authentication against the REST service happens in two steps:

1. Requesting an authentication token by calling the service named /authentication/credentials
2. Requesting arbitrary services thereafter and sending this token for subsequent API calls.

A typical request body looks like this:

```
{
        "realm": <STRING|UNDEFINED>,
        "type": <STRING>,
        "login": <STRING|UNDEFINED>,
        "key": <STRING>,
        "persistent": <BOOLEAN|UNDEFINED>
}
```

- The realm property is optional and will be read from the sub-domain by the backend most of the time. Setting it is required when the REST service is accessed from a different domain, what might be relevant for mobile devices.
- The property type could contain one of the following strings: credentials or token.
- The properties key and login contain login information.

A typical response body looks like this:

```
{
		"data": {
			"key": <STRING>
		}
		...
}
```

- The key property contains the actual authentication token, that has to be sent we each subsequent API call (if non-public resources are accessed)

Search Result Filtering: Introducing Filter and MappingResource

General Filter Process

For all resources, that have the filter URL query parameter documented, the following server-side filter process *can* be applied:

1. Client sends GET request for a list of resources with some filters set (which are of type Filter)
2. For a successful request, the server sends
    1. the effective, filtered resource list (in the result field of the data property) and
    2. an array of MappingResource (in the filter field of the data property) that may be used as filters *on this result list's resources* in subsequent API calls
3. Optional step: Client may send new request filtered according to resource specific filters (e.g. for even finer grained resource filtering)

Bidirectional Nature of Filter

Although implicitly depicted in the "General Filter Process" above, note how the Filter concept applies for both:

1. Requests being sent *to* the server and
2. responses being sent *from* the server.

However, the main difference is:

- When requesting a service, Filters are defined as a filter URL query parameter
- When consuming a server response, Filters are transmitted in the filter field of the server's response body (on the same level as the requested fields of the queried resource)

MappingResource in Filter

Maps a filter property to the corresponding resource field.

Field	Type	Description
property	<STRING>	Name of the filter property.
value	see filter type	Type of the associated resource field.

An example request looks like this:

```
filter=[{
        "property": "id",
        "value": "10cc93453cf232d8013cf264cccd007a"
     }, {
          "property": "name",
          "value": "a name"
    }]
```

Filter Types

Date Filter

Date Filter are used to filter the search result for date based fields. The value can be a predefined value key or a dynamic date expression. See Predefined filter values the for more information about available predefined value keys.

The date expression is used to filter the search result with a date range. The syntax of the date pattern is:

SAB_PATTERN: <FROM Date Expression> TO <TO Date Expression>

The prefix to mark the value as a date expression is SAB_PATTERN. The expression starts with an "anchor" date, which can be either NOW or a date string. It can then follow by a math expression, supporting +, - and / (rounding). The units supported are

- YEAR
- MONTH
- WEEK
- DAY
- HOUR
- MINUTE
- SECOND
- MILLI (milli second)
- * (unlimited future or past date)

Examples for the maths expressions:

- +1h - add one hour
- -1d - subtract one day
- /d - round down to the nearest day

Examples

- Last 30 days until today (including exact time, for example: today, 15:41:32): SAB_PATTERN:NOW-30DAY TO *
- Last 30 days until today (rounded to end of day): SAB_PATTERN:NOW/DAY-30DAY TO NOW/DAY+0DAY-1MILLI
- From today to 30 days in future: SAB_PATTERN:NOW TO NOW+30DAY

```
{
  "property": "created",
  "value": "SAB_PATTERN:NOW TO NOW+30DAY"
}
```

Term Filter

Term Filter are used to filter the search result on string based fields. The value must be a single term. Whitespace is not possible. A common usage for Term Filter is filtering for ids or tags. Only search items are contained in search result which match the Term Filter. The Term Filter can also be negated, that means only search items are contained in search result which don’t match the filter. To negate the filter property simply add a leading - character.

Examples

Only include search items for a given content view:

- Field: branchIds
- Value: ID of content view (In this example: 4374ef464668853c014694590b175831)

```
{
  "property": "elements.branchIds",
  "value": "4374ef464668853c014694590b175831"
}
```

Only include search items with a tag “foobar”:

- Field: tags
- Value: foobar

```
{
  "property": "elements.tags",
  "value": "foobar"
}
```

Exclude search items with are of type “pdf”:

- Field: type
- Value: pdf

```
{
  "property": "-elements.tags",
  "value": "pdf"
}
```

Predefined filter values

The predefined value keys can be used to filter the result with a simple keyword. This is an easy way to filter the search with common filters. The syntax of predefined filter values are:

`{short}<predefined value key>`

Example

```
{
  "property": "created",
  "value": "{short}today"
}
```

Filter Properties

Created Date

Field can be used to filter the search result by created timestamp.

Field: created

Predefined value key	Description
today	Shows only content, which has been created today. Today means, from 00:00 in the beginning of the day until now.
last_week	Shows only content, which has been created in the last week, starting by now -7 Days at 0:00 o’clock.
last_two_weeks	Shows only content, which has been created in the last two week, starting by now -14 Days at 0:00 o’clock.
last_month	Shows only content, which has been created in the last month, starting at 0:00 o’clock.
older	Shows only content, which has a created timestamp older than a month.

Example:

```
{
   "property" : "averageRating",
   "value" : "{short}last_month"
}
```

Last Modification Date

Field can be used to filter the search result by the last modification time.

Field: lastModified

Predefined value key	**Description**
today	Shows only content, which has been modified today. Today means, from 00:00 in the beginning of the day until now.
last_week	Shows only content, which has been modified in the last week, starting by now -7 Days at 0:00 o’clock.
last_two_weeks	Shows only content, which has been modified in the two last week, starting by now -14 Days at 0:00 o’clock.
last_month	Shows only content, which has been modified in the two last month, starting at 0:00 o’clock.
older	Shows only content, which has a modified timestamp older than a month.

Example:

```
{
  "property" : "lastModified",
  "value" : "{short}last_month"
}
```

Average Rating filter

Field can be used to filter the search for content with a given average rating range.

Field: averageRating

Predefined value key	Description
excellent	Shows only content, which has an average rating between 4.5 and 5.1.
good	Shows only content, which has an average rating between 3.5 and 4.49.
average	Shows only content, which has an average rating between 2.5 and 3.49.
bad	Shows only content, which has an average rating between 1.5 and 2.49.
awful	Shows only content, which has an average rating between 1 and 1.49.
none	Shows only content, without an average rating.

Example:

```
{
  "property" : "averageRating",
  "value" : "excellent"
}
```

Write permission filter

Field can be used to filter the search for content which is created by current user or for content which can be edited by current user.

Field: writePermission

Predefined value key	Description
my_content	Shows only content, which is created by current user.
write_permission	Shows only content, which could be modified by current user.

Example:

```
{
   "property" : "writePermission",
   "value" : "{short}my_content"
}
```

Valid to Time

Field can be used to filter the search result for expiring content.

Field: `elements.validTo`

Predefined value key	**Description**
today	Shows only content, which has become invalid today. Today means, from 00:00 in the beginning of the day until now.
week	Shows only content, which has become invalid in the next week, starting by now -7 Days at 0:00 o'clock.
two_weeks	Shows only content, which has become invalid in the next two week, starting by now -14 Days at 0:00 o'clock.
month	Shows only content, which has become invalid in the next month, starting at 0:00 o'clock.
later	Shows only content, which has become invalid in the next later then a month.
unlimited	Shows only content, which has no configured valid to time.

Example:

```
{
   "property" : "elements.validTo",
   "value" : "{short}today"
}
```

Valid from Time

Field can be used to filter the search items for content that is not valid, yet.

Field: elements.validFrom

Predefined value key	Description
today	Shows only content, which has become valid today. Today means, from 00:00 in the beginning of the day until now.
week	Shows only content, which will become valid in the next week, starting by now -7 Days at 0:00 o’clock.
two_weeks	Shows only content, which will become valid in the next week, starting by now -14 Days at 0:00 o’clock.
month	Shows only content, which will become valid in the next month, starting at 0:00 o’clock.
later	Shows only content, which will become valid after the next month.

Example:

```
{
  "property" : "elements.validFrom",
  "value" : "{short}today"
}
```

Tags

Field can be used to filter the search items for content that contains the given tags.

Field: elements.tags

Example:

```
{
  "property" : "elements.tags",
  "value" : "myTag"
}
```

View

Field can be used to filter the search items for content that is assigned to the given view id.

Field: elements.branchIds

Example:

```
{
   "property" : "elements.branchIds",
   "value" : "viewId"
}
```

Document Type

Field can be used to filter the search items for content of the given type

Field: type

Example:

Possible values are file endings of uploaded documents or images, but also SABIO specific elements like Sabio-Texts, Sabio-News, …

```
[
   {
      "property" : "type",
      "value" : "pdf"
   },
   {
      "property" : "type",
      "value" : "text"
   }
]
```

Resource

Field can be used to filter the search items for specific resources. Resources are views, users, reports, texts, documents and a lot of others.

Field: resource

Example:

```
[
  {
    "property" : "resource",
    "value" : "branch"
  },
  {
    "property" : "resource",
    "value" : "user"
  }
]
```

Search Fields

Common Search Fields

The common search fields can be used for filtering or sorting in every resource, which is included in search. Currently the following resources are searchable: text, document, submission and message.

Field	Type	Filter Type	Can be sorted	Description
id	String	none	no	Unique ID of entity. Can be used to load the entity itself
created	Date	Date Filter	yes	Timestamp of entity's creation date. See Introduction and Basic Concepts details on date format.
lastModified	Date	Date Filter	yes	Timestamp of entity's last modification date. See Introduction and Basic Concepts for details on date format.
validTo	Date	Date Filter	yes	Date until when the resource is valid. Has to be an RFC822 formatted date string. See Introduction and Basic Concepts for details on date format.
validFrom	Date	Date Filter	yes	Date from when the resource is valid. Has to be an RFC822 formatted date string. See Introduction and Basic Concepts for details on date format.
branchIds	String[]	Term Filter	no	Array of all ids of branches, which are assigned to this entity. Can be used to filter all content for a given branch
lastModifiedById	String	Term Filter	yes	ID of user who modified the resource.
resource	String	Term Filter	yes	Discriminator for the type of the SABIO resource that corresponds to the search item. Legal values are text, message, document and submission

Field	Type	Filter Type	Can be sorted	Description
type	String	Term Filter	yes	Discriminator for the binary data type of the indexed resource. Only applies to SABIO Document resources.
tags	String[]	Term Filter	no	Array of keywords used to tag this resource.
writePermission	String	Predefined Filter	no	A field to filter the search for content owned by the current user or for content, which can be modified by the current user.

Changelog

- v2.0.6 (SABIO Knowledge 5.16)
    - cleaned up section session-key and api-key
    - added section changelog
    - added menu Deprecations
- v2.0.5 (SABIO Knowledge 5.16)
    - added sabio-client to section Authentication, Token and ApiKey
- v2.0.4 (SABIO Knowledge 5.16)
    - added search field
    - added search filtering
- v2.0.2 (SABIO Knowledge 5.16)
    - deprecated /token/login
    - cleaned up section session-key and api-key
    - added How to use to section api-key
- v2.0.1 (SABIO Knowledge 5.15)
    - added How to use to section session-key

Overview

This chapter describes each service in detail, giving documentation about

1. Whether access to the service is protected or public,
2. which methods and paths are available,
3. which URL query parameters are consumed,
4. which Filter can be applied to a resource list returned by the service
5. and which fields can be requested and/or sent and how to use them.

Authentication

URL	**/authentication/credentials**
Access	public
Methods	POST, DELETE

Service for authenticating users against SABIO. On success, an authentication token is returned, contained in an UserAuthResponse.

This service supports three types of authentication:

1. With credentials,
2. by an already existing token
3. by an api-key created via api-key service

In general, the generated token has to be send as sabio-auth-token header in each request that requires a user. In some cases the token needs to be appended as a sabio-auth-token query parameter (e.g. when downloading documents). Also, each client has to add a sabio-client header with the name of it's client type for each request it makes. This client type string has to be added to a (comma separated list) named Settings/System/Key for multiple log-in (Side note: Our SABIO web client always sends a header like this for REST calls sabio-client: {"name":"SABIO 5","version":"1.23.0"}).

```
curl --request GET \
  --url 'https://mycompany.sabio.de/sabio/services/user/profile' \
  --header 'sabio-auth-token: 1jefzdq4yq2i8urp4zccavwvs1hrawpta3f9etr1ix1brax9ie' \
  --header 'sabio-client: {"name":"MyCompanyApp","version":"1.2.3"}'
```

Available Paths

Method	**Path Segment**	**Returned Value**	**Description**
POST	/authentication/credentials	single	Consumes an UserAuthRequest and returns an UserAuthResponse. On authentication success, UserAuthResponse contains a valid authentication token.

Response Format

```
{
  "data": {
    "key": <STRING>
  }
  ...
}
```

- See section *"Resource Fields"* for documentation about properties
    - key

Resource Fields

UserAuthRequest

Name	Type	Required for POST	Description
type	String	yes	Authentication type. Possible values are credentials or token.
login	String	special	Represents the user's login name when type is set to credentials. In this case, this field is mandatory for POST requests!
key	String	yes	Value depends on type field: If type is token it contains an authentication token or api-key, if type is credentials it contains the user's password.
realm	String	yes	Identifier of the realm, the user lives in. This property will be auto-detected from the request URL if omitted.
persistent	Boolean	no	Indicates, whether the server should create a persistent token.

UserAuthResponse

Name	Type	Description
key	String	The token to authenticate with for subsequent service calls.

Name	Type	Description
user	LightUserResource	The authenticated user.

Api-key

URL	**/api-key**
Access	protected
Methods	POST, PUT, DELETE, GET

Service for creating an api-key to authenticate requests against SABIO. On success, an api-key as token is returned.

Security hints

The api-keys should be created only with validTo dates. Renew the keys every 30 days or more often.

Users assigned to api-keys should not be able to access critical services like create user, edit roles, ….

Available Paths

Method	**Path Segment**	**Protected by Role**	**Returned Value**	**Description**
POST	/api-key	APIKEY_CREATE	single	Creates an ApiKey, on success, response contains a valid api-key token.
PUT	/api-key/__UUID__	APIKEY_UPDATE	single	Updates an existing ApiKey with given Id.
DELETE	/api-key/__UUID__	APIKEY_DELETE	single	Deletes an existing ApiKey with given Id.
GET	/api-key	APIKEY_READ	List	Returns a list of ApiKeys.
GET	/api-key/__UUID__	APIKEY_READ	single	Returns an ApiKey with given Id if exists.

Response Format

```
{
  "data": {
      "id": <STRING>,
      "name": <STRING>,
      "token": <STRING>,
      "userId": <STRING>,
      "login": <STRING>
   }
   ...
}
```

- See section *“Resource Fields”* for documentation about properties
  - id
  - name
  - token
  - userId
  - login
  - validTo

Resource Fields

Name	**Type**	**Read-Only**	**Required for POST**	**Description**
id	String	yes	no	Id of the api-key.
name	String	no	yes	Name of the api-key.
token	String	yes	no	Token of the api-key.
userId	String	no	yes (when login empty)	ID of the user assigned to this api-key.
login	String	no	yes (when id empty)	Login of the user assigned to this api-key. (since 5.16)
validTo	Date	no	yes	End date of the validity period of this api-key. See for details on date format.

How to create an api-key

The following section demonstrates how an api-key is created step by step. The created api-key is assigned to user 4nils and valid until 20 July 2017 10am. To perform this example, you need an SABIO user with admin-rights (CREATE_ROLE, CREATE_USER) and curl. Also, all users have to be on the same realm.

Create role Tokencreator

First create a new role. This role contains only the required rights to create api-keys for any users.

1. Login as admin
2. Go to the settings tab and click on Add user role
3. Set name to Tokencreator
4. Select all rights in the API Keys section
5. Click on Save

Create user Tokengenerator

Second create a new user. This user is only to create api-keys for any users.

1. Login as admin
2. Go to the settings tab and click on Add user
3. Set Firstname, Lastname, Language and Email
4. Set Login name to Tokengenerator
5. Set Password to s3cr3t
6. Select all groups
7. Remove all roles
8. Select role Tokencreator
9. Click on Save

Authenticate user Tokengenerator

Third authenticate user Tokengenerator against SABIO. The example is using the credentials method, but you can use any method you want. The point is, that you get an authentication token back to make further requests as user Tokengenerator.

The login property is the login of the user Tokengenerator, the key property is the password of this user and the realm property is qa-test.

```
curl --request POST \
  --url "https://mycompany.sabio.de/sabio/services/authentication/credentials" \
  --header 'sabio-client: {"name":"MyCompanyApp","version":"1.2.3"}' \
  --header 'Content-Type: application/json' \
  --data '{"login":"Tokengenerator","key":"s3cr3t","realm":"qa-test"}'
```

The resulting JSON contains a key property. This is the authentication token assigned to user Tokengenerator.

```
{
  "data": {
    "key": "vbemy9kt36t4vbeo1q7xxut0sobuh5vezbxuus067j29pr1v",
    ...
  }
  ...
}
```

Create an api-key

Fourth create an api-key for user 4nils. The value of the header attribute sabio-auth-token is the authentication token of user Tokengenerator created in the previous step. The login property is the login of the user, the api-key will be assigned to (in this case 4nils) and the validTo property is the end date of the validity period for this api-key. The start date of the validity period is now and not configurable.

```
curl --request POST \
   --url "https://mycompany.sabio.de/sabio/services/api-key" \
   --header 'sabio-client: {"name":"MyCompanyApp","version":"1.2.3"}' \
   --header 'sabio-auth-token: vbemy9kt36t4vbeo1q7xxut0sobuh5vezbxuus067j29pr1v' \
   --header 'Content-Type: application/json' \
   --data '{"login":"4nils", "validTo":"Mon Jul 24 2017 10:00:00 GMT+0200"}'
```

The resulting JSON contains a token property. The value of this property is the generated authentication token assigned to this api-key.

```
{
  "data": {
    result : {
      "token": "1jefzdq4yq2i8urp4zccavwvs1hrawpta3f9etr1ix1brax9ie",
      ...
    }
  }
  ...
}
```

Now you can execute requests as user 4nils.

```
curl --request GET \
   --url "https://mycompany.sabio.de/sabio/services/user/profile" \
   --header 'sabio-client: {"name":"MyCompanyApp","version":"1.2.3"}' \
   --header 'sabio-auth-token: 1jefzdq4yq2i8urp4zccavwvs1hrawpta3f9etr1ix1brax9ie'
```

Note: The value of the header attribute sabio-auth-token is the authentication token of user 4nils created in the previous step.

Tree

URL	**/tree**
Access	protected
Methods	POST, GET, PUT, DELETE

Service for managing SABIO TreeResources. Can be used to load single nodes or complete trees.

A TreeResource is a hierarchical object representing a node in the tree, containing all its child-TreeResources.

Available Paths

Method	Path Segment	Protected by Role	Returned Value	Description
GET	/tree/0	TREE_READ	single	Returns the root tree with all child nodes, visible for the user.
GET	/tree/__UUID__	TREE_READ	single	Returns one specific TreeResource, identified by its UUID.
POST	/tree	TREE_CREATE	single	Returns the newly created TreeResource.
PUT	/tree/__UUID__	TREE_UPDATE	single	Returns the modified TreeResource, identified by its UUID.
DELETE	/tree/__UUID__	TREE_DELETE	single	Deletes the TreeResource identified by its UUID. Only a Status is returned.

Response Format

```
{
  "data": {
    "result": <TreeResource>
  },
  ...
}
```

URL Query Parameters

Name	Type	Methods	Description
filter	MappingObject[]	GET	A Filter to reduce the returned result set. For concrete available filters, see corresponding section below.

Available Filters

The following values are valid for the filter query parameter:

Name	Type	Description
depths	Number	The number of nested children to load. Default value is set to 0. For unlimited depths this needs to be set to -1.
branchIds	UUID[]	The UUIDs of views to load.

Resource Fields

Name	Type	Read-Only	Required for POST	Description
objectType	String	yes	yes	Identifier of this resource's type. Fixed value is 'TreeResource'.
title	String	no	yes	Title of this resource.
description	String	no	no	Detailed description of this resource.
createdBy	LightUserResource	no	yes	The user who created the resource.
group	LightGroupResource	no	yes	A user group that has access to this resource.

Name	Type	Read-Only	Required for POST	Description
validFrom	Date	no	no	Date from when the resource is valid. See for details on date format.
validTo	Date	no	no	Date until when the resource is valid. See Introduction and Basic Concepts for details on date format.
children	Tree[]	yes	no	An array of sub nodes.
attachments	ResourceReference[]	yes	no	An array of available attachments. *Currently only text resources can be added as attachments.*
tags	String[]	no	no	Array of keywords used to tag this resource.
parentId	String	no	yes	The UUID of this tree node's parent tree node. Is null for the root tree.
branches	LightViewResource[]	no	yes	Array of branches, that restrict which resources can be attached to this tree. Only resources - /text resources in particular - that *have a subset of the views (previously known as branches) the tree has* can be attached to it.

Text

URL	**/text**
Access	protected
Methods	GET, PUT, POST, DELETE

Service for managing SABIO TextResources.

Available Paths

Method	Path Segment	Protected by Role	Returned Value	Description
GET	/text/__UUID__	TEXT_READ	single	Returns one specific TextResource, identified by its UUID.
POST	/text	TEXT_CREATE	single	Returns the newly created TextResource.
PUT	/text/__UUID__	TEXT_UPDATE	single	Returns the modified TextResource, identified by its UUID.
DELETE	/text/__UUID__	TEXT_DELETE	single	Deletes the TextResource identified by its UUID. Only a Status is returned.

Response Format

```
{
  "data": {
    "result": <TextResource>
  },
  ...
}
```

URL Query Parameters

None.

Resource Fields

Name	Type	Read-Only	Required for POST	Description
objectType	String	yes	yes	Identifier of this resource's type.
title	String	no	yes	Title of this resource.
createdBy	LightUserResource	no	yes	The user who created the resource.
group	LightGroupResource	no	yes	User group assigned to this resource.
validFrom	Date	no	no	Date from when the resource is valid. See Introduction and Basic Concepts for details on date format.
validTo	Date	no	no	Date until when the resource is valid. See for details on date format.
paths	LightTreeNodeResource[][]	no	yes	A two-dimensional array of LightTreeNodeResources, representing the multiple possible paths to this resource in the tree.
fragments	TextFragmentResource[]	no	yes	An array of TextFragmentResources,

Name	Type	Read-Only	Required for POST	Description
				defining the actual content of this TextResource.

Sub-Resource TextFragmentResource

The TextFragmentResource is a sub-entity used to represent the actual content of a TextResource.

A TextFragmentResource *cannot be directly accessed via URL*, that is, there intentionally exists no explicit service for TextFragmentResources.

Resource Fields

Name	Type	Read-Only	Required for POST	Description
objectType	String	yes	yes	Identifier of this resource's type.
content	String	no	yes	The actual content as HTML.
branches	LightViewResource[]	no	yes	Array of views (previously known as branches) which restrict which usergroups are allowed to see the text.
tags	String[]	no	no	Array of keywords used to tag this resource.
attachments	LightDocumentResource[]	yes	no	An array of available attachments (LightDocumentResource, see chapter *Standard resources*).
submissionId	UUID	no	no	UUID of the submission belonging to this TextFragmentResource.

Name	Type	Read-Only	Required for POST	Description
contextValues	ContextValue[]	no	no	Values for a context type. A context type is associated to a resource type and can be optional or mandatory. If mandatory it must be set.

Document

URL	**/document**
Access	protected
Methods	GET, PUT, POST, DELETE

Service for managing SABIO DocumentResources.

Available Paths

Method	**Path Segment**	**Protected by Role**	**Returned Value**	**Description**
GET	/document/__UUID__	DOCUMENT_READ	single	Returns one specific DocumentResource, identified by its UUID.
POST	/document	DOCUMENT_CREATE	single	Returns the newly created DocumentResource.
PUT	/document/__UUID__	DOCUMENT_UPDATE	single	Returns the modified DocumentResource, identified by its UUID.
DELETE	/document/__UUID__	DOCUMENT_DELETE	single	Deletes the DocumentResource identified by its UUID. Only a Status is returned.

Response Format

```
{
  "data": {
    "result": <DocumentResource>
  },
 ...
}
```

URL Query Parameters

None.

Resource Fields

Name	Type	Read-Only	Required for POST	Description
objectType	String	yes	yes	Identifier of this resource's type.
title	String	no	yes	Title of this resource.
description	String	no	no	Detailed description of this resource.
token	FileTokenResource	no	yes	FileTokenResource identifying the temporary uploaded file in the file system.
fileName	String	no	yes	Name of the associated document in the file system.
tags	String[]	no	no	Array of keywords used to tag this resource.
links	TextElementDocumentMappingResource[]	no	yes	A one-dimensional array of TextElementDocumentMappingResources, representing the multiple possible "text elements" this document is linked to.
createdBy	LightUserResource	no	yes	The user who created the resource.
group	LightGroupResource	no	yes	User group assigned to this resource.

Name	Type	Read-Only	Required for POST	Description
validFrom	Date	no	no	Date from when the resource is valid. See for details on date format.
validTo	Date	no	no	Date until when the resource is valid. See Introduction and Basic Concepts for details on date format.
branches	LightViewResource[]	no	yes	Array of views (previously known as branches) which restrict which usergroups are allowed to see the text.

FileTokenResource

See chapter for service

Sub-Resource TextElementDocumentMappingResource

The TextElementDocumentMappingResource is a sub-entity used to link the document to a TextFragmentResource the actual content of a TextResource.

A TextElementDocumentMappingResource *cannot be directly accessed via URL*, that is, there intentionally exists no explicit service for TextElementDocumentMappingResources.

Resource Fields

Name	Type	Read-Only	Required for POST	Description
objectType	String	yes	yes	Identifier of this resource's type. static value: 'TextElementDocumentMappingResource'
id	String	no	yes	UUID of a TextFragmentResourceTextFragmentResource the document is or should be linked to.

Files

URL	**/files**
Access	protected
Methods	POST

Interface to manage SABIO file resources. As files are binary data and thereby not presentable in JSON format. However, when you upload a new file, a JSON string is send with a Content-Type header set to text/plain to confirm success. This construct is required to be able to upload a file from within an iframe. Temporary created files also contain a token for later usage. The token may be used by other resources (e.g. documents) to reference the file.

To create a file, the request needs to be send as multipart.

The sabio-auth-token needs to be set as query parameter.

Available Paths

Method	Path Segment	Protected by Role	Returned Value	Description
POST	/files	DOCUMENT_CREATE	single	Returns the newly created FileResource.

Multipart Format

Name	Type	Writable	Required	Description
file	String	yes	yes	Contains the file data.

Response Format

```
{
  "data": {
    "result": <FileTokenResource>
  },
  ...
}
```

URL Query Parameters

Name	Type	Methods	Available for paths	Description
sabio-auth-token	String	POST	/files	authentication token

Additional Header to use for posting a file Accept:text/html, Content Type set to multipart/form-data

FileTokenResource

Name	Type	Read-Only	Required for POST	Description
objectType	String	yes	yes	Identifier of this resource's type.
token	String	yes	yes	token of uploaded attachment.

Message / News

URL	/message
Access	protected
Methods	GET, PUT, POST, DELETE

Service for managing SABIO NewsResources.

Throughout the whole section, treat "news" as a synonym for "message".

Available Paths

Method	Path Segment	Protected by Role	Returned Value	Description
GET	/message/__UUID__	MESSAGE_READ	single	Returns one specific NewsResource, identified by its UUID.
POST	/message	MESSAGE_CREATE	single	Returns the newly created NewsResource.
PUT	/message/confirm/__UUID__	MESSAGE_READ	single	Marks the NewsResource identified by UUID as confirmed and returns it.
DELETE	/message/__UUID__	MESSAGE_DELETE	single	Deletes the NewsResource identified by its UUID. Only a Status is returned.

Response Format

```
{
  "data": {
    "result": <NewsResource>
  },
  ...
}
```

URL Query Parameters

None.

Resource Fields

Name	Type	Read-Only	Required for POST	Description
objectType	String	yes	yes	Identifier of this resource’s type.
title	String	no	yes	Title of this resource.
content	String	no	yes	The NewsResource’s text content.
targetGroups	LightGroupResource[]	no	yes	An array of GroupResources that defines the user groups that will receive this news *in SABIO internally*.
validFrom	Date	no	no	Date from when the resource is valid. See Introduction and Basic Concepts for details on date format.
validTo	Date	no	no	Date until when the resource is valid. See for details on date format.
contextValues	ContextValue[]	no	no	Values for a context type. A context type is associated to a resource type and can be optional or mandatory. If mandatory it must be set.

Submission

URL	**/submission**
Access	protected
Methods	GET, PUT, POST

Service for managing SABIO SubmissionResources. A SubmissionResource represents a user created submission concerning a certain TextResource.

Available Paths

Method	Path Segment	Protected by Role	Returned Value	Description
GET	/submission/__UUID__	SUBMISSION_READ	single	Returns one specific SubmissionResource, identified by its UUID.
POST	/submission	SUBMISSION_CREATE	single	Returns the newly created SubmissionResource.
PUT	/submission/__UUID__	SUBMISSION_UPDATE	single	Returns the modified SubmissionResource, identified by its UUID.

Response Format

```
{
   "data": {
      "result": <SubmissionResource>
   },
   ...
}
```

URL Query Parameters

None.

Resource Fields

Name	Type	Read-Only	Required for POST	Description
objectType	String	yes	yes	Identifier of this resource’s type. Fixed value is ‘SubmissionResource’.
title	String	yes	yes	Title of this resource.
targetId	UUID	yes	yes	UUID of the resource, the SubmissionResource is created for.
targetResource	String	yes	yes	Identifier of the type of the resource, the SubmissionResource is created for. At the moment, only text is supported.
comments	SubmissionCommentResource[]	no	no	An array of comments, associated with this SubmissionResource. *Important:* At the moment, it is only possible to add comments to a SubmissionResource, but not to modify existing comments!
status	String	yes	no	Identifier of the status of this SubmissionResource. Valid values are closed, inprogress and pending.

Sub-Resource SubmissionCommentResource

The SubmissionCommentResource is a sub-entity used to represent a comment of a SubmissionResource.

A SubmissionCommentResource *cannot be directly accessed via URL*, that is, there intentionally exists no explicit service for SubmissionCommentResources.

Resource Fields

Name	Type	Read-Only	Required for POST	Description
id	UUID	yes	no	The resources UUID. Automatically assigned on creation.
text	String	no	yes	The comment’s content
created	Date	yes	no	The creation date of this resources. See Introduction and Basic Concepts for details on date format.
createdBy	LightUserResource	yes	no	The user who created the resource.

Message / News

URL	**/message**
Access	protected
Methods	GET, PUT, POST, DELETE

Service for managing SABIO NewsResources.

Throughout the whole section, treat “news” as a synonym for “message”.

Available Paths

Method	**Path Segment**	**Protected by Role**	**Returned Value**	**Description**
GET	/message/__UUID__	MESSAGE_READ	single	Returns one specific NewsResource, identified by its UUID.
POST	/message	MESSAGE_CREATE	single	Returns the newly created NewsResource.
PUT	/message/confirm/__UUID__	MESSAGE_READ	single	Marks the NewsResource identified by UUID as confirmed and returns it.
DELETE	/message/__UUID__	MESSAGE_DELETE	single	Deletes the NewsResource identified by its UUID. Only a Status is returned.

Response Format

```
{
  "data": {
    "result": <NewsResource>
  },
  ...
}
```

URL Query Parameters

None.

Resource Fields

Name	Type	Read-Only	Required for POST	Description
objectType	String	yes	yes	Identifier of this resource’s type.
title	String	no	yes	Title of this resource.
content	String	no	yes	The NewsResource’s text content.
targetGroups	LightGroupResource[]	no	yes	An array of GroupResources that defines the user groups that will receive this news *in SABIO internally*.
validFrom	Date	no	no	Date from when the resource is valid. See Introduction and Basic Concepts for details on date format.
validTo	Date	no	no	Date until when the resource is valid. See for details on date format.
contextValues	ContextValue[]	no	no	Values for a context type. A context type is associated to a resource type and can be optional or mandatory. If mandatory it must be set.

Search

URL	**/search**
Access	protected
Methods	GET

Service for performing searches within SABIO. Results only contain contents that are accessible for the current user.

Available Paths

Method	Path Segment	Protected by Role	Returned Value	Description
GET	/search	SEARCH_READ	list	Returns a *non-generic* SearchResultResource.
GET	/search/suggest	SEARCH_READ	list	Returns a list of AutoCompleteResources, which are suggested based on query parameter q.

Response Format

- total contains the total number of available results for the fired request (in general, larger result sets are paginated)
- The properties limit and start are values of applied *"URL Query Parameters"* (see below)
- See section *"Resource Fields"* for documentation about property filter
- The property autoReSearch contains a boolean value indicating if an auto re-search is executed. An auto re-search is executed if the search does not find any result for the given query. In this case the Search API determines an alternative query and executes this query.
- The property queryTerm contains the query term for the effectively executed search. In the case when autoReSearch is false it contains the submitted query, otherwise it contains the alternative query term.
- The property originalQueryTerm contains the query term which is submitted with the query. It is null in the case when the search returns results for the submitted query.

Path /search

```
{
  "data": {
    "result": <NULL>|<SearchResultResource>[],
    "total": <INTEGER>,
    "limit": <INTEGER>,
    "start": <INTEGER>,
    "filter": <MappingObject>[],
    "autoReSearch" : <BOOLEAN>,
    "queryTerm" : <STRING>,
    "originalQueryTerm" : <STRING>
  },
  ...
}
```

Path /search/suggest

```
{
  "data": {
    "result": <NULL>|<AutoCompleteResource>[],
    "total": <INTEGER>,
    "limit": <INTEGER>,
    "start": <INTEGER>
  },
  ...
}
```

URL Query Parameters

Name	Type	Methods	Available for paths	Description
filter	MappingObject[]	GET	/search, /search/suggest	A Filter to reduce the returned result set. For concrete available filters, see Search Result Filtering.
filterList	String	GET	/search	A comma separated list of resource

Name	Type	Methods	Available for paths	Description
				properties. This list is used to request facets for given property list.
q	String	GET	/search, /search/suggest	A search query string.
start	Integer	GET	/search, /search/suggest	Index of the fist returned resource within the request's result set, starting with 0.
limit	Integer	GET	/search, /search/suggest	Number of resources effectively returned from the request's result set. Interpret as "result chunk size".

Resource Fields

SearchResultResource

Name	Type	Description
resource	String	Resource type.
title	String	Title of this resource.
id	UUID	UUID identifying this resource.
excerpt	String	An excerpt of the search result item, in the form on a HTML fragment.

Name	Type	Description
authorId	UUID	The UUID of the user who has created or is the current owner of the indexed resource.
branches	Object[]	An array of branches, the indexed resource is associated with.
validFrom	Date	Date from when the resource is valid. See Introduction and Basic Concepts for details on date format.
validTo	Date	Date until when the resource is valid. See for details on date format.
lastModifiedById	UUID	The UUID of the user who modified the resource the last time.

AutoCompleteResource

Name	Type	Description
text	String	The suggestion’s text.
count	Integer	Number of expected results.

Session-key (supported until August 2018 - please use /api-key instead)

URL	**/token/login**
Access	protected
Methods	GET

Service for creating a session-key to authenticate requests against SABIO. A session-key is assigned to a user and a user may have multiple session-keys at the same time (Mehrfachanmeldung). On success, a session-key ̀is created, valid for hour and returned to the caller.

Available Paths

Method	Path Segment	Protected by Role	Returned Value	Description
GET	/token/login/__LOGIN__	USER_CAN_CAPTURE_OTHER_USER	single	Returns a TokenResource.

Response Format

```
{
  "data": {
    "result": {
      "token": <STRING>
    }
  }
  ...
}
```

Resource Fields (TokenResource)

Name	Type	Read-Only	Required for POST	Description
token	String	yes	no	Authentication token of this session-key.

How to create an session-key

The following section demonstrates how a session-key is created step by step. The created session-key is assigned to user 4nils and valid for one hour. To perform this example, you need an SABIO user with admin-rights (CREATE_ROLE, CREATE_USER) and curl. Also, all users have to be on the same realm.

Create role Tokencreator

First create a new role. This role contains only the required rights to create session-keys for other users.

1. Login as user with admin-rights
2. Go to the settings tab and click on Add user role
3. Set name to Tokencreator
4. Select May create token for other user in the User section
5. Click on Save

Create user Tokengenerator

Second create a new user. This user is only to create session-keys for other users.

1. Login as user with admin-rights
2. Go to the settings tab and click on Add user
3. Set Firstname, Lastname, Language, Email
4. Set Login name to Tokengenerator
5. Set Password to s3cr3t
6. Select all groups
7. Remove all roles
8. Select role Tokencreator
9. Click on Save

Add sabio-client MyCompanyApp

Third add your client application to the sabio-client list. This is necessary because the session-key will be assigned to a given user and sabio-client. Each time a session-key is created, the existing session-key will be overwritten. If the sabio-client header attribute is empty, the default name will be used. The default name (unknown) and the name of the SABIO web client (SABIO 5) do not have to be added explicitly.

1. Login as user with admin-rights
2. Go to the settings tab and click on Settings
3. Select System and go to the text field labeled with Key for multiple log-in
4. Add to the comma separated list the name of your client (e.g. MyCompanyApp)
5. Click on Save

Authenticate user Tokengenerator

Fourth authenticate user Tokengenerator against SABIO. The example is using the credentials method, but you can use any method you want. The point is, that you get an authentication token back to make further requests as user Tokengenerator.

The login property is the login of the user Tokengenerator, the key property is the password of this user and the realm property is my. There is also the header attribute sabio-client with the name and version of your application. This client type string has to be added for each request (Side note: Our SABIO web client always sends a header like this for REST calls sabio-client: {"name":"SABIO 5", "version":"1.23.0"}).

```
curl --request POST \
  --url "https://mycompany.sabio.de/sabio/services/authentication/credentials" \
  --header 'sabio-client: {"name":"MyCompanyApp","version":"1.2.3"}' \
  --header 'Content-Type: application/json' \
  --data '{"login":"Tokengenerator","key":"s3cr3t","realm":"my"}'
```

The resulting JSON contains a key property. This property is the authentication token assigned to user Tokengenerator.

```
{
  "data": {
    "key": "s5056vzx288ptocv0t9s9ug6rhlmid6krbz6s4p20wbfrol1",
    ...
  }
  ...
}
```

Create a session-key

Fifth create a session-key for user 4nils. The value of the header attribute sabio-auth-token is the authentication token of user Tokengenerator created in the previous step. The value of the header attribute sabio-client is the name and version of your application . The last parameter of the path is the login of the user. The session-key will be assigned to to the given user and sabio-client (in this case 4nils and MyCompanyApp).

```
curl --request GET \
   --url "https://mycompany.sabio.de/sabio/services/token/login/4nils" \
   --header 'sabio-client: {"name":"MyCompanyApp","version":"1.2.3"}' \
   --header 'sabio-auth-token: qxg0o98bzy2et4hsskwia14msknawx2fsxtkqxtdjibgwm5w'
```

The resulting JSON contains a token property. The value of this property is the generated authentication token assigned to this session-key.

```
{
  "data": {
    result : {
            "token": "1jefzdq4yq2i8urp4zccavwvs1hrawpta3f9etr1ix1brax9ie",
            ...
    }
  }
```

```
   ...
}
```

Now you can execute requests as user 4nils.

```
curl --request GET \
    --url "https://mycompany.sabio.de/sabio/services/user/profile" \
    --header 'sabio-client: {"name":"MyCompanyApp","version":"1.2.3"}' \
    --header 'sabio-auth-token: 1jefzdq4yq2i8urp4zccavwvs1hrawpta3f9etr1ix1brax9ie'
```

Note: The value of the header attribute sabio-auth-token is the authentication token of user 4nils created in the previous step.