

# ELO REST API

ELO REST API

# Table of contents

<b>ELO REST API</b>	<b>4</b>
<b>Overview</b>	<b>5</b>
Interactive UI	5
Specification	5
Scope	6
Design	6
CORS	7
<b>Authentication</b>	<b>8</b>
Backend apps authentication	8
"External" Browser apps - login/logout	
"Internal" Browser apps - sharing the Web Client session	
<b>Search</b>	<b>10</b>
Search "anywhere"	
Search by keywording	11
<b>Files</b>	<b>12</b>
Get a folder	12
Get a document	12
Performance	13
Downloading a document	14
Listing files in a folder	14
Creating a folder	15
Uploading a new document using an URL	15
Uploading a new document in the browser	16
Uploading a new document *version*	
Moving a file	17
Updating the short description	17
Getting or updating metadata	18
<b>Metadata</b>	<b>19</b>
Reading metadata/keywording	
Updating metadata/keywording	
Change mask	20
Change mask and metadata/keywording	
<b>Members</b>	<b>21</b>
List members of a group	21
Get user	21
Add new user	22

---

Set user's group memberships

## Misc

**24**

---

Calling a registered function (IX)

## Masks (technical name for metadata forms)

---

Get mask information

**25**

# ELO REST API

# Overview

## Interactive UI

It is highly recommended to experiment with the interactive specification by opening the browser at the following URL:

`http(s)://<elo-server>/rest-<repo>/doc`



### AUTHENTICATION

Type	Authentication Procedure
<b>http: basic</b>	Send 'Authorization' in header which will contains the word 'Basic ' followed by a space and a base64-encoded string username:password. <div> <input type="text" value="username"/> <input type="text" value="password"/> <input type="button" value="SET"/> </div>
<b>apiKey:</b>	Send 'ticket' in 'header' with the given value <div> <input type="text" value="api-token"/> <input type="button" value="SET"/> </div>

### /API/FILES

Api Files

<input type="button" value="GET"/>	/api/files	Files.byPath
<input type="button" value="POST"/>	/api/files	Files.add
<input type="button" value="PATCH"/>	/api/files	Files.set
<input type="button" value="GET"/>	/api/files/{id}	Files.get
<input type="button" value="DELETE"/>	/api/files/{id}	Files.delete
<input type="button" value="GET"/>	/api/files/{id}/acl	Files.getACL
<input type="button" value="GET"/>	/api/files/{id}/children	Files.getChildren

You will see the list of endpoints and can try them out directly by sending requests and receiving live responses.

## Specification

The specification for the REST API is self generated and follows the *OpenAPI 3* standard. Sometimes it is useful to access the raw JSON specification, for example if used by some tools. This raw JSON specification is available at:

`http(s)://<elo-server>/rest-<repo>/v3/api-docs`

## Scope

The goal of this API is *not* to cover all the IX's functionality. Rather, it is to offer a simple way to perform common operations.

The API covers the following areas:

- files
  - basic information
  - download/upload
  - metadata (*also known as "keywording"*)
    - IX fields
    - map fields
  - child entries (*for directories*)
  - versions (*for documents*)
  - ACL (*read-only*)
- basic search
  - by fulltext
  - by metadata
- members
  - users
  - groups
- system
  - masks
  - colors
  - registered functions (IX)

## Design

Most endpoints follow a similar pattern:

- GET `/api/.../{id}`
  - Gets an item.
  - Both `id` or `guid` can be used in the URL path.
- POST `/api/...`
  - Adds a new item.
  - The returned value will be of the form `{"id":..., "guid":..., "name":...}`.
- PATCH `/api/...`
  - Updates an item.
  - The `id` or `guid` *must* be present in the body.
- DELETE `/api/.../{id}`
  - Deletes an item.
  - Both `id` or `guid` can be used in the URL path.

For every endpoint containing `{id}` *in the URL*, both `id` or `guid` can be used interchangeably. Sometimes, where the IX internally allows it, the identifying "name" can be used too. This applies for example for user names, mask names, color names...

## CORS

If the web app is not hosted on the same Tomcat, setting up CORS will be necessary to enable remote websites to access the API.

Instead of configuring CORS for the whole Tomcat, it can be set explicitly for the REST module.

In the `config.properties`, simply add:

```
allowedOrigins = https://example.com, https://another.net
```

Using the `*` wildcard is possible but should only be enabled during development/testing. It is highly discouraged in production because of obvious security concerns.

# Authentication

## Backend apps authentication

All API endpoints require authentication, this can be either done using:

- *Basic authentication* with the ELO user name/password
- a valid *ticket* in either of:
  - the query (adding &ticket=...)
  - a "ticket" header
  - a "ticket" cookie

## "External" Browser apps - login/logout

Instead of providing authentication each time, it is also possible to login directly to establish a session between the browser and the API.

Browser sessions are disabled by default. To enable them, edit the configuration located at:

<elo-install-path>/config/rest-.../<server-name>/config.properties

Update it with following values:

```
allowSessions = true
allowedOrigins = https://example.com, https://another.net
```

Then restart the REST API using the Tomcat Manager.

*Note that for security reasons, allowSessions is not compatible with the "wildcard" origin (allowedOrigins = \*). If you want to enable browser sessions externally, you have to explicitly list the allowed origins.*

Once properly configured, users can login:

POST /login

```
{
  "username": "Alice",
  "password": "secret",
  "lang": "en"
}
```

This session follows the usual rules of the Tomcat. It typically expires after 10 minutes of inactivity.



The `lang` parameter specifies the language to connect with, which is the two-letter code of any of the supported client languages. This influences translation keys and error messages.

To logout, invoke:

POST `/logout`

## "Internal" Browser apps - sharing the Web Client session

If you invoke the API through `https://.../ix-MyRepo/plugin/de.elo.ix.plugin.proxy/rest/api/...`, the session of the Web Client will be shared and authentication will be automatically performed.

In other words, you do not have to provide credentials or call `/login` by using such URLs if the user is already logged in using the Web Client (or another ELO web application).

Note that this solution is *not* compatible with cross origin use cases. This only works if the web app is configured to run behind the IX Proxy Plugin, like "ELO Apps".

Alternatively, you could deploy a classic web app on the Tomcat, and edit the IX Proxy Plugin configuration located at:

```
<elo-install-path>/config/ix-.../<server-name>/de.elo.ix.plugin.proxy.properties
```

There, you can add a forwarding URL to your web app:

```
yourapp=http(s)://any-valid-url
```

After restarting the IX, all requests to `https://.../ix-MyRepo/plugin/de.elo.ix.plugin.proxy/yourapp/...` would be forwarded to your web app URL and calls to the REST API would share the Web Client session.

# Search

## Search "anywhere"

This example searches all documents and folders containing "elo" anywhere: the name, the document's content or the associated metadata/keywording.

Request:

GET /api/search?words=elo

Response:

```
[
  {
    "guid": "(25A7C0C1-DA8C-176C-50EC-778215E6D69C)",
    "id": 31,
    "name": "elo.profile",
    "type": 3,
    "isDir": true,
    "desc": "",
    "lock": "",
    "ownerName": "ELO Service",
    "access": "RWD-LP",
    "parentId": 30,
    "dateArchived": "2020-10-26T10:09:00Z",
    "dateCustom": null,
    "dateModified": "2020-10-26T09:09:44Z"
  },
  ...
  {
    "guid": "(C1F90D1D-2C4B-D03E-475E-3C8ADCB26AD4)",
    "id": 40,
    "name": "ELO Service",
    "type": 23,
    "isDir": true,
    "desc": "",
    "lock": "",
    "ownerName": "ELO Service",
    "access": "RWD-LP",
    "parentId": 28,
    "dateArchived": "2020-10-26T10:09:00Z",
    "dateCustom": null,
    "dateModified": "2020-11-12T16:14:02Z"
  }
]
```

```
}  
]
```

It is possible to restrict this, for example by using one of the following:

```
GET /api/search?words=elo&where=TITLE
```

```
GET /api/search?words=elo&where=DOCUMENT
```

```
GET /api/search?words=elo&where=KEYWORDING
```

Please take into consideration that the response is fixed to at most 1000 results (without any possibility to alter it). This implies that responses can be rather large for vague terms. It also is not suited to walk through all the documents of a repository. It is best used with precise search terms.

## Search by keywording

Search all documents of category ABC regarding contract XYZ.

```
POST /api/search/keywording
```

```
{  
  "CATEGORY": "ABC",  
  "CONTRACT_ID": "XYZ"  
}
```

Like the previous search, it will return the list of found documents and folders.

Note that you currently cannot specify the mask you are looking for. This feature might come in a future version. Also, in ELO, only "index fields" can be searched, not "map" fields.

# Files

## Get a folder

Gets all information about /Administration/EL0apps/Icons.

The following requests are equivalent:

```
GET /api/files?path=/Administration/EL0apps/Icons
```

```
GET /api/files/86
```

```
GET /api/files/(8615AC76-DBE4-4907-03E5-F974F6C9F13A)
```

The response contains all information about the folder:

```
{
  "info": {
    "guid": "(8615AC76-DBE4-4907-03E5-F974F6C9F13A)",
    "id": 86,
    "name": "Icons",
    "type": 3,
    "isDir": true,
    "desc": "",
    "lock": "",
    "ownerName": "Administrator",
    "access": "RWD-LP",
    "parentId": 73,
    "dateArchived": "2020-10-26T10:09:00Z",
    "dateCustom": null,
    "dateModified": "2020-10-26T09:09:55Z"
  },
  "acl": [...],
  "children": [...],
  "keywording": {...},
  "versions": null,
  "content": null
}
```

## Get a document

Gets all information about /Administration/EL0apps/Icons/tile-Add.ico.

The following requests are equivalent:

```
GET /api/files?path=/Administration/EL0apps/Icons/  
GET /api/files/126  
GET /api/files/(7E2739B6-3EE5-8296-7C35-C07500621A8A)
```

The response contains all information about the document:

```
{  
  "info": {  
    "guid": "(7E2739B6-3EE5-8296-7C35-C07500621A8A)",  
    "id": 126,  
    "name": "tile-Add.ico",  
    "type": 284,  
    "isDir": false,  
    "desc": "",  
    "lock": "",  
    "ownerName": "ELO Service",  
    "access": "RWDE-P",  
    "parentId": 86,  
    "dateArchived": "2020-10-26T10:09:00Z",  
    "dateCustom": null,  
    "dateModified": "2020-10-26T09:09:57Z"  
  },  
  "acl": [...],  
  "children": [],  
  "keywording": {...},  
  "versions": [...],  
  "content": {...}  
}
```

## Performance

In both previous examples, all the information about the document/directory is retrieved.

For folders, this includes the list of all child files, and for documents, this includes the list of versions and basic information about the content.

Naturally, such an operation is more resource consuming than retrieving only what is actually needed.

Instead of retrieving everything using:

```
GET /api/files/{id}
```

It is also possible to only get the subset of interest:

```
GET /api/files/{id}/acl
```

```
GET /api/files/{id}/children
```

```
GET /api/files/{id}/content (*)
GET /api/files/{id}/info
GET /api/files/{id}/keywording
GET /api/files/{id}/versions
```

This reduces traffic and is more efficient. Like the other endpoints, both `id` and `guid` can be used interchangeably.

(\*) Calling `GET /api/files/{id}/content` actually downloaded the document instead of providing content metadata in the ELO 12 Beta Version. It was deprecated in REST API v20.02 and might be replaced by the content metadata in ELO 21.

## Downloading a document

This example downloads the "tile-Add.ico" icon previously listed.

The following requests are equivalent:

```
GET /api/files/126/download
GET /api/files/(7E2739B6-3EE5-8296-7C35-C07500621A8A)/download
```

Response:

<binary-data> of the tile-Add.ico file

In REST API prior to v20.02, `GET /api/files/{id}/content` should be used to download the document.

## Listing files in a folder

This example lists the files in the "/Administration/ELOapps/Icons" folder previously obtained.

The following requests are equivalent:

```
GET /api/files/86/children
GET /api/files/(8615AC76-DBE4-4907-03E5-F974F6C9F13A)/children
```

Response:

```
[
  {
    "guid": "(7E2739B6-3EE5-8296-7C35-C07500621A8A)",
    "id": 126,
    "name": "tile-Add.ico",
    "type": 284,
    "isDir": false,
    "desc": "",
    "lock": ""
```

```

    "ownerName": "ELO Service",
    "access": "RWDE-P"
  },
  ...
  {
    "guid": "(DE239939-F00C-69C2-9AF0-89C7E72D3519)",
    "id": 238,
    "name": "tile-WorldMap.ico",
    "type": 284,
    "isDir": false,
    "desc": "",
    "lock": "",
    "ownerName": "ELO Service",
    "access": "RWDE-P"
  }
]

```

## Creating a folder

Request:

POST /api/files

```

{
  "info": {
    "name": "Example Folder",
    "parentId": 123,
  }
}

```

Response:

```

{
  "id": 1234,
  "guid": "(...)",
  "name": "Example Folder"
}

```

## Uploading a new document using an URL

The following example will upload the page <https://www.wikipedia.org> in ELO

Request:

POST /api/files

```
{
  "content": {
    "filename": "index.html",
    "url": "https://www.wikipedia.org"
  },
  "info": {
    "name": "Wikipedia Homepage",
    "parentId": 123,
  }
}
```

Response:

```
{
  "id": 1234,
  "guid": "...",
  "name": "Wikipedia Homepage"
}
```

Notes:

- the filename extension determines the document type and icon in ELO
- do not provide an id nor guid since it is a new file
- many properties are read-only and setting them has no effect
- the guid and name in the response are only in version 20+

## Uploading a new document in the browser

This example shows a HTML snippet able to upload a document from the browser.

The following example assumes that the user is already authenticated. Please consult the chapter Authentication for further information on how to achieve this in browser use cases.

```
<form id="sampleUpload">
  <input type="text" name="name">
  <input type="file" name="file">
  <input type="text" name="versionComment">
  <input type="submit">
</form>

<script>
  formElem.onSubmit = async (e) => {
    e.preventDefault();
```



```
var parentId = 123;
let response = await fetch('.../api/files/' + parentId, {
  method: 'POST',
  body: new FormData(formElem)
});
let result = await response.json();
alert("Uploaded document id: " + result.id);
};
</script>
```

## Uploading a new document *version*

Uploading a new version of an existing document can only be done using the POST `/api/files/{id-or-guid}/content` endpoint and is similar to the previous examples.

## Moving a file

This example moves the file 126 to the folder `/Administration` (having the id 2)

Moving a file to another folder is actually the same as updating its `parentId`.

Request:

PATCH `/api/files/126/info`

```
{
  "parentId": 2
}
```

In this case, the `parentId` should be the id of the target parent folder and a `guid` is not allowed.

## Updating the short description

Request:

PATCH `/api/files/126/info`

```
{
  "desc": "Here is an updated short description!"
}
```

Here as well, the `guid` could be used instead.

## Getting or updating metadata

---

See Metadata

# Metadata

## Reading metadata/keywording

Request:

GET /api/files/{id-or-guid}/keywording

Response:

```
{
  "maskId": 2,
  "maskNameOriginal": "E-mail",
  "fields": {
    "EL00UTL1": "example@noreply",
    "EL00UTL2": "something else...",
    ...
  },
  "map": {
    "FOO": "BAR"
  }
}
```

Note that the metadata is always key/value strings. Even if the field is defined as number, date or something else, the value is always handled as a string.

## Updating metadata/keywording

Request:

PATCH /api/files/{id-or-guid}/keywording

```
{
  "fields": {
    "EL00UTL1": "pseudomail@noreply"
  },
  "map": {
    "FOO": "BAR"
  }
}
```

This will only update the provided fields, and leave the others unchanged.

Note that metadata are always key/value strings. Even if the field is defined as a number, date or something else, a string should always be provided.

## Change mask

Request:

PATCH /api/files/{id-or-guid}/keywording

```
{
  "maskNameOriginal": "E-Mail"
}
```

or

PATCH /api/files/{id-or-guid}/keywording

```
{
  "maskId": 2
}
```

## Change mask and metadata/keywording

Both operations can be combined into one.

Request:

PATCH /api/files/{id-or-guid}/keywording

```
{
  "maskNameOriginal": "E-Mail",
  "fields": {
    "EL00UTL1": "pseudomail@noreply"
  },
  "map": {
    "FOO": "BAR"
  }
}
```

Note that setting `maskId` or `maskNameOriginal` will not only update the mask (if necessary) but it will also *reset all other index fields*. Therefore, if you want to just update a few fields without affecting the others, omit `maskId` and `maskNameOriginal` in the request.

# Members

## List members of a group

The following requests are equivalent:

GET /api/members/groups/Administrators

GET /api/members/groups/(E10E1000-E100-E100-E100-E10E10E10E43)

GET /api/members/groups/9998

Response:

```
{
  "guid": "(E10E1000-E100-E100-E100-E10E10E10E43)",
  "id": 9998,
  "name": "Administrators",
  "parentGroups": [],
  "childGroups": [],
  "users": [
    {
      "guid": "(E10E1000-E100-E100-E100-E10E10E10E40)",
      "id": 0,
      "name": "Administrator"
    },
    ...
  ],
  "rights": null
}
```

## Get user

The following requests are equivalent:

GET /api/members/users/Alice

GET /api/members/users/(4357EA87-9744-B3A5-6911-4414A5160288)

GET /api/members/users/5

Response:

```
{
  "guid": "(4357EA87-9744-B3A5-6911-4414A5160288)",
  "id": 5,
  "name": "Alice",
  "password": null,
}
```

```
"change_password": null,  
"email": "alice@wonder.lands",  
"nologin": false,  
"online": false,  
"memberOf": [  
  {  
    "guid": "(E10E1000-E100-E100-E100-E10E10E10E43)",  
    "id": 9998,  
    "name": "Administrators"  
  },  
  ...  
]  
}
```

## Add new user

Request:

POST /api/members/users

```
{  
  "name": "Alice",  
  "password": "secret",  
  "change_password": "MUST_CHANGE",  
  "email": "alice@wonder.lands",  
  "memberOf": [  
    {  
      "id": 9998,  
      "name": "Administrators"  
    }  
  ]  
}
```

Response:

```
{  
  "guid": "(4357EA87-9744-B3A5-6911-4414A5160288)",  
  "id": 5,  
  "name": "Alice"  
}
```

Note that you can directly provide the groups the user should belong to. In order to do this, you must provide the groups id to identify the groups in `memberOf`. Both `guid` and `name` are ignored in this case.

You cannot set individual rights/permissions. It is good practice that users inherit rights from the groups they are member of.

## Set user's group memberships

The following request for Alice sets "Teamroom Users" as the only group she is a member of.

Request:

PATCH /api/members/users

```
{
  "guid": "(4357EA87-9744-B3A5-6911-4414A5160288)",
  "id": 5,
  "name": "Alice",
  "memberOf": [
    {
      "guid": "(C0553F55-AEF0-0C9C-5166-78F28047D0FD)",
      "id": 2,
      "name": "Teamroom Creators"
    }
  ]
}
```

Either the id or guid of the user is required for this request.

Similar to when adding a user, you must provide the groups id to identify the groups in memberOf. Both guid and name are ignored in this case.

# Misc

## Calling a registered function (IX)

This request will call the registered function "RF\_sol\_function\_ChangeColor" to change the color of the ELO "Administration" folder to red.

Request:

POST /api/misc/functions/RF\_sol\_function\_ChangeColor

```
{  
  "objId": 2,  
  "color": "red"  
}
```

Please refer to the business solution documentation for more information about the specific remote functions available depending on your system and installed modules.



# Masks (technical name for metadata forms)

## Get mask information

The following requests are equivalent:

```
GET /api/system/masks/(E10E1000-E100-E100-E100-E10E10E10E32)
```

```
GET /api/system/masks/2
```

```
GET /api/system/masks/E-mail
```

Response:

```
{
  "id": 2,
  "guid": "(E10E1000-E100-E100-E100-E10E10E10E32)",
  "originalName": "E-mail",
  "forDocs": true,
  "forFolders": false,
  "forSearch": true,
  "isUsed": true,
  "fields": [
    {
      "id": 0,
      "key": "ELOOUTL1",
      "type": 3000,
      "label": "From",
      "isRequired": false,
      "isHidden": false
    },
    ...
    {
      "id": 6,
      "key": "ELOOUTLREF",
      "type": 3000,
      "label": "Reference",
      "isRequired": false,
      "isHidden": false
    }
  ]
}
```

The type is a internal constant referring to the field's type (text, number, date...).

Note that even if the field is defined as a number, date or something else, it is nevertheless always handled as a string (of max length 255).