Mind42 API Documentation

Introduction

Mind42 offers an oAuth2 interface that generally should behave like Google's oAuth API. Therefore it's a good start to read Googles documentation as a start:
https://developers.google.com/accounts/docs/OAuth2

Authentication & general technical details

The base URL of Mind42s oAuth2 API is: https://mind42.com/api/oauth2

Currently there are two scopes for the Mind42 API: "read" and "write". A possible third scope ("account" to access user account actions) might be added in the future.

To start the process, as described in the Google documentation, let the users authorize your application by sending them to /api/oauth2/auth with the described parameters (response_type, client_id, redirect_uri, scope, state).

Like the Google oAuth implementation also Mind42 offers a redirect_uri value of "urn:ietf:wg:oauth:2.0:oob".

On this page the user has to sign in, and is asked whether he or she wants to grant the requested scope to your application. If he or she does, the configured redirect will be called with an authorization code. This code is valid for 1 minute and can only be used once.

This authorization code can then be exchanged for an access and a refresh token. The endpoint for this is /api/oauth2/token. As described in the Google documentation this is a POST containing the values code, client_id, client_secret, (the same) redirect_uri as before and grant_type. When exchanging the authorization code for tokens the grant_type has to be "authorization_code".

With the received accessToken you can then do API calls. The access token is valid for 60 minutes and will simply stop working after this time (as indicated in the "expires_in" response when getting the token).

But you also receive a refreshToken. This token is valid until the user revokes the grant of your application to access his or her user account. With the refresh token you can get a new access token, which you then can use to continue using the API. Unlike Google, Mind42 will currently always issue a refresh token.

Using the refresh token works exactly like exchanging the authorization_code for tokens, but this time you provide the refresh token instead of the authorization code. The only difference is, that the POST to /api/oauth2/token doesn't have to contain a redirect_uri and the grant_type must be "refresh_token".

To actually use the API then, call the below listed API endpoints with the access token as "Authorization: Bearer" HTTP header or access_token query parameter, as described in the Google documentation.

Generally all API calls will return a JSON string as response. The base form of this response is:

```
{
        code: 200 (OPTIONAL)
        error: "error message" (OPTIONAL)
        data: JSON (OPTIONAL)
}
```

In case an error occurs when calling /api/oauth2/token, only the "error" field will be set in the response.
In case an error occurs when calling an API endpoint, the "error" message will be set, the code will be 500 (server error) or some other code specific to the API call, and data will be null.
In case everything succeeds code will be 200, error will be null, and data will be set to the JSON response as specified in the API call.

Generally all requests are HTTP GET requests. Only methods where bigger data (like mind map JSON, or a mind map description) has to be sent to the server HTTP POST requests will be used.

API methods

Mindmap methods

mindmapList

URL: /api/oauth2/v1/mindmapList
HTTP method: GET
Scope: read
Parameters:

- detailed: BOOLEAN

Returns:
Simple Response

```
[
      {
            mindmapId: UUID,
            name: STRING
      },
      ...
]
```

Detailed Response

```
[
      {
            mindmapId: UUID,
            name: STRING,
            role: ROLECODE,
            dateEdit: TIMESTAMP,
            dateCreation: TIMESTAMP,
            published: BOOLEAN,
            publishedPublic: BOOLEAN,
            collaborators: [
                  {
                        userId: UUID,
                        name: STRING,
                        role: ROLECODE
                  },
                  ...
            ],
            mindmapGroups: [
                  UUID,
                  ...
            ]
      },
      ...
]
```

Explanation:
This call returns an array of mind maps in the users account. There is one parameter:

- **detailed:** If true, the detailed response as shown above will be sent, otherwise just the mind map ids and names will be returned.

Each array element is an object with the following properties:

- **mindmapId:** The id of the mind map.
- **name:** The mind map name. It is currently taken automatically from the label of the root node of the mind map.
- **role:** The role the users has in this mind map. "cr" if the user is the creator, "ed"/"vi" if he or she's just an invited collaborator.
- **dateEdit:** Timestamp of the last change to the mind map.
- **dateCreation:** Timestamp of the creation date of the mind map.
- **published:** Whether the mind map is published (publicly available read only view) of the mind map.
- **publishedPublic:** Only relevant if mind map is published. If true then the mind map will also be listed in the Mind42 Gallery.
- **collaborators:** Array of all users of the mind map (including own user). For each user the id, name and role is supplied.
- **groups:** Array of group UUIDs. In their Mind42 account users can assign the mind maps into groups (like the GMail tagging system). Each group has a UUID. This array contains all the groups UUIDs this mind map is assigned to.

mindmapGet
URL: /api/oauth2/v1/mindmapGet
HTTP method: GET

Scope: read
Parameters:

- mindmapId: UUID
- online: BOOLEAN

Returns:

```
{
      mindmapId: UUID,
      revisionId: NUMBER,
      sessionId: UUID, (OPTIONAL)
      name: STRING,
      role: ROLECODE,
      dateCreation: TIMESTAMP,
      dateEdit: TIMESTAMP,
      published: BOOLEAN,
      publishedPublic: BOOLEAN,
      collaborators: [
            {
                  userId: UUID,
                  name: STRING,
                  role: ROLECODE
            },
            ...
      ],
      root: MINDMAPROOTJSON
}
```

Explanation:
This call retrieves the JSON of a Mind42 mind map. It has two parameters:

- **mindmapId:** The UUID of the mind map you want to load.
- **online:** If true, starts an online editing session.

Regarding the online flag: If false, you will only get the mind map data and nothing else will happen. If true, the server will start an online editing session for this mind map. This means that the user will appear as "currently" online, and other collaborators will see him or her. In this case you'll receive a sessionId in the response, which you can use for the sessionDelta API method to participate in an online collaboration. The online session expires automatically after some time when you don't use the sessionDelta API call.

The return value is very similar to each line of the mindmapList call:

- **mindmapId:** The id of the mind map.
- **revisionId:** The id of the loaded revision. Unlike all other IDs, this is no UUID, but a simple number.
- **sessionID:** Only set if the request parameter “online” was true.
- **name:** The mind map name.
- **role:** The role the user has in this mind map.
- **dateCreation:** Timestamp of the creation date of the mind map.
- **dateEdit:** Timestamp of the last change to the mind map – and therefore actually the timestamp when this revision was created.
- **published:** Whether the mind map is published

- **publishedPublic:** If public, whether the mind map is listed in the Mind42 public gallery.
- **collaborators:** Array of all users of the mind map (including own user). For each user the id, name and role is supplied.
- **root:** The actual mind map data JSON (the root node).

mindmapSave

URL: /api/oauth2/v1/mindmapSave
HTTP method: POST
Scope: write
Parameters:

- mindmapId: UUID
- revisionId: NUMBER
- overwriteToken: STRING (OPTIONAL)
- root: MINDMAPROOTJSON

Returns:

```
{
	saved: BOOLEAN,
	revisionId: NUMBER, (OPTIONAL)
	overwriteToken: STRING (OPTIONAL)
}
```

Explanation:
This call can be used to send a whole mind map to Mind42 to save it. This is not the mode Mind42 usually works in, as in the online version changes are sent incrementally. But e.g. for offline use this call can be used. It has the following parameters:

- **mindmapId:** The UUID of the mind map you want to save.
- **revisionId:** The revision id your changes are based on (the last revision id you got when loading the map).
- **overwriteToken:** An optional parameter that you can supply to enforce to overwrite the mind map with your JSON, although it was changed on the server in the meantime. More about this below.
- **root:** The mind map JSON.

The return values of the call are as follows:

- **saved:** A boolean, whether the map was saved or not. If it is false, it means that there was a revisionId conflict (the map has changed on the server in the meantime), and an overwriteToken will be present.
- **revisionId:** Only exists if “saved” is true. Contains the revisionId of the newly created revision.
- **overwriteToken:** A string that will only be returned if there was a conflict on save.

The idea behind the overwrite token is this. If you use the Mind42 API in an offline way, you'll usually get the mind map when online, edit it when offline, and try to save the changed JSON when online again. In the meantime though, the map mind might have been changed on the server.

This is detected using the revisionId. With every save call you'll have to send a revisionId upon which your changes are based on. It this revisionId doesn't match with the last known revisionId on the server, it's treated as a conflict. In case of a conflict you have two options. Either you insist on overwriting the server JSON with your version of the JSON, or you save the changed JSON as a new mind map. You might delegate this decision to the user, using a dialog which asks something like "The mind map you are trying to save already has been changed on the server. Would you like to 1) save it anyways and overwrite the server version or 2) save the changed mind map as a new mind map.". In case a new mind map should be created you can simply save the mind map JSON using the mindmapCreate call. In case you want to overwrite it, call the mindmapSave method again, but this time with the overwriteToken you received. If the correct overwrite token is supplied, the save call will succeed. However, also if supplying an overwrite token you just received, the save call could fail in case the mind map already changed again since receiving the previous overwrite token. In this case the failed call will include a new overwrite token.

mindmapCreate

URL: /api/oauth2/v1/mindmapCreate
HTTP method: POST
Scope: write
Parameters:

- root: MINDMAPROOTJSON

Returns:

```
UUID
```

Explanation:
This call is used to create a new mind map on Mind42. It only has one parameter:

- **root:** The mind map JSON.

As can be seen, like mindmapSave, also mindmapCreate takes any mind map data. So create an empty new mind map, simply send the JSON for a root node, with some label for the root node (text attribute). The name of the mind map is automatically taken form the root node label. But this also allows you to create mind maps with more than the default root node (e.g. imported mind maps, ...).

The call simply returns the UUID if the created mind map on the server. So the “data” attribute of the response JSON will simply be the string of the UUID.

mindmapDelete

URL: /api/oauth2/v1/mindmapDelete
HTTP method: GET

Scope: write
Parameters:

- mindmapId: UUID

Returns:

```
true
```

Explanation:
This call is used to delete an existing mind map on Mind42. Is has one parameter:

- **mindmapId:** The mind map UUID.

If successful the mind map will be deleted from the user account. It can't be restored. The return value is simply true – so that "data" attribute of the response JSON will simply be true.

mindmapThumbnail

URL: /api/oauth2/v1/mindmapThumbnail
HTTP method: GET
Scope: read
Parameters:

- mindmapId: UUID
- size: STRING (OPTIONAL "thumbnail" or "gallery")

Returns:

```
PNGIMAGE
```

Explanation:
This URL can be used to get the PNG image of a mind map. It has two parameters:

- **mindmapId:** The mind map UUID.
- **size:** The size of the thumbnail.

Mind map thumbnails can only be accessed for mind maps the user has access to (own mind maps, mind maps where the user is a collaborator or public mind maps). Two sizes of thumbnails are available: 130x90 (size=thumbnail) and 200x170 (size=gallery). If no size is given it defaults to "thumbnail" and therefore 130x90 pixels.

Mindmap Group methods

mindmapGroupList

URL: /api/oauth2/v1/mindmapGroupList
HTTP method: GET

Scope: read
Parameters: NONE
Returns:

```
[
	{
		mindmapGroupId: UUID,
		name: STRING
	},
	...
]
```

Explanation:
This call returns an array of mind map groups the user created. Each group simply has a name and an id, and the user can assign one or more mind map groups to every mind map to organize them.

mindmapGroupCreate

URL: /api/oauth2/v1/mindmapGroupCreate
HTTP method: POST
Scope: write
Parameters:

- name: STRING

Returns:

```
UUID
```

Explanation:
This call creates a new mindmap group with the given name. The response is simply the UUID of the newly created MindmapGroup (like e.g. mindmapCreate).

mindmapGroupDelete

URL: /api/oauth2/v1/mindmapGroupDelete
HTTP method: GET
Scope: write
Parameters:

- mindmapGroupId: UUID

Returns:

```
true
```

Explanation:
This call removes a mind map group with the given UUID. If the call succeeds it simply returns true.

mindmapGroupRename

URL: /api/oauth2/v1/mindmapGroupRename
HTTP method: POST
Scope: write

Parameters:

- mindmapGroupId: UUID
- name: STRING

Returns:

```
true
```

Explanation:
This call simply renames the mindmap group with the given mindmapGroupId and sets the new given name. If the call succeeds the response simply is true.

mindmapGroupAssign

URL: /api/oauth2/v1/mindmapGroupDelete
HTTP method: POST
Scope: write
Parameters:

- mindmapId: UUID
- mindmapGroupIds: ARRAY of UUIDs: [UUID, UUID, ...]

Returns:

```
true
```

Explanation:
Assigns the given mind map groups to the given mind map. Mind map groups are per user – therefore every user has his own mind map groups, and all mindmapGroup calls (including this one) can be executed by every collaboration role (viewer, editor, creator). mindmapGroupAssign replaces all current group assignments and does not add the given groups the the already existing assigned groups. So to remove all group assignments from a mindmap simply use this call with an empty array. If the call succeeds it simply returns true.

User methods

userInfo

URL: /api/oauth2/v1/userInfo
HTTP method: GET
Scope: read
Parameters:

- userId: UUID (OPTIONAL)
- mindmapId: UUID (OPTIONAL)

Returns:

```
{
      userId: UUID,
      name: STRING
}
```

Explanation:
This call can be used to get more info about a user based on the userId (currently only the username). This is useful for getting the user info of the currently logged in user, or if a new collaborator joins an editing session, to get the name of the given userId.
Of course you can't simply get the user info for every user on Mind42, but only the user info from users you are collaborating with. So here's the explanation of the parameters:

- **userId:** If no userId is given, your own user info is returned. So without any parameters, this call can be used to get info about the currently logged in user (the user you have the accessToken for).
- **mindmapId:** If you provide a userId other than your own, you'll also have to provide the mind map id this users is collaborating on with you. So Mind42 will always check whether the given userId is actually a collaborator of the given mindmapId, and only then will return the user info.

The return values of the call are as follows:

- **userId:** The userId.
- **name:** The name of the user. This is not necessarily the "username" used to sign in to Mind42, but can be a (non unique) display name defined by the user. In case the username is an email address and no display name is set, the part after the "@" will be replaced with "…" for privacy reasons.

Session methods

sessionDelta

URL: /api/oauth2/v1/sessionDelta
HTTP method: POST
Scope: read (and write for sending deltas)
Parameters:

- sessionId: UUID
- deltas: MINDMAPDELTAJSON (OPTIONAL)

Returns:

```
{
      revisionId: NUMBER/NULL,
      deltas: MINDMAPDELTAJSON,
      users: [
            UUID,
            ...
      ],
      refresh: STRING (OPTIONAL)
}
```

Explanation:
This call is the heart of the Mind42 collaboration system. Once you started an online editing session (mindmapGet with parameter online=true) and received a sessionId, you'll have to regularly call this API method to keep the online editing session alive.

"Being" online means, that other collaborating users of the same mind map see you as being online right now. When you don't call sessionDelta regularly you will be shown as offline to other users after about 30 seconds. But the sessionId is not immediately invalidated when this happens. You can still use the same sessionId within 30 minutes since it's last use. When doing so Mind42 will show the user as back online again.

So to stay online, and be an active part of a collaboration session you should "ping" the API with the sessionId every few seconds. More on this later.

But this call is also used to transmit changes to the mind map from the client to the server and from the server to the client. If you have write scope, you can send changes to the server using the deltas array. The response from the server will also contain a deltas array containing all changes from the server.

- **sessionId:** The sessionId you received by mindmapGet.
- **deltas:** A list of deltas that you want to transmit to the server.

The return values of the call are as follows:

- **revisionId:** The most recent revisionId (including the changes you sent to the server and the ones received form the server). This can be NULL in case "refresh" is set.
- **deltas:** A list of deltas that represent the changes from the server since you last sessionDelta call.
- **users:** An array of user UUIDs that are "online" right now. You can use this to detect whether other users are online right now, and show them to the user (e.g. using the userInfo API call)
- **refresh:** A string which is only present in case the session was invalidated by a revision restore or mindmap overwrite. If the owner of the mind map restores an older revision, or the last regular revision got overwritten with a completely new version (e.g. user mindmapSave with an overwriteToken), you'll have to start a new online editing session.

So basically you have to use this call for multiple reasons: To keep the online editing session alive (even if no other collaborator is online and no change was made on the client). We call this a “ping” and recommend to do this every 20 seconds. If there are collaborators online, and you don’t have any changes to send, you still want to get the changes the others made. We call this a deltaRead and recommend doing it every 5-10 seconds. Both scenarios mentioned don’t set the deltas parameter and are therefore possible with the read scope and with every collaboration role (creator, editor, viewer). The third reason is to push changes you made to the server. This is only possible with the write scope, and only if you have the creator or editor role in this mind map collaboration (viewers can’t change anything on the mind map). In case you have changes to send to the server you should do this immediately (and bypass the regular 5-10 second interval).

sessionEnd

URL: /api/oauth2/v1/sessionEnd
HTTP method: GET
Scope: read
Parameters:

- sessionId: UUID

Returns:

```
true
```

Explanation:
As explained before a session will time out automatically, and other collaborators will see that a user is no longer online if that happens. If possible however you can call the sessionEnd call to end a session immediately. The paramters are:

- **sessionId:** The sessionId you received by mindmapGet.

If the call is successful it will simply return true.

Revision methods

revisionList

URL: /api/oauth2/v1/revisionList
HTTP method: GET
Scope: read
Parameters:

- mindmapId: UUID

Returns:

```
[
	{
		revisionId: NUMBER,
		timestamp: TIMESTAMP,
		age: NUMBER
	},
	...
]
```
Explanation:
This call returns a list of all revisions (current and past versions) of a mind map. The Mind42 server creates a new revision with every change (sessionDelta). But not every revision is kept. Generally 1 revision is kept for every 5 minutes. Every day though a cleanup process is removing some revisions, and keeps less and less revisions the older the mind map is. As an example: For the last week 1 revision per 10 minutes is kept, for the next 3 weeks 1 revision per hour is kept, for the next 5 months 1 revision per day is kept and so on. To get the revision list you have to supply the following parameters:

- **mindmapId:** The UUID of the mind map you want to fetch the revision list for.

The response is an array (sorted by revision age from oldest to newest), which holds an object for every revision. The properties of these revision objects are:

- **revisionId:** The numeric id of the revision (as received by mindmapGet for example)
- **timestamp:** The UNIX timestamp of the creation date of this revision.
- **age:** The age of the revision in seconds based on the current time. Since the time zone handling of the timestamp is not really working yet this is the best way to show how old a revision is.

revisionGet
URL: /api/oauth2/v1/revisionGet
HTTP method: GET
Scope: read
Parameters:
- mindmapId: UUID
- revisionId: NUMBER

Returns:
```
{
	mindmapId: UUID,
	revisionId: UUID,
	userId: UUID,
	timestamp: TIMESTAMP,
	root: MINDMAPROOTJSON
}
```
Explanation:
This call is related to mindmapGet, but only returns a specifc revision without all the mind map meta data. Together with revisionList this can be

used to browse and show old revisions of a mind map. The parameters are:

- **mindmapId:** The UUID of the mind map you want to fetch a revision for.
- **revisionId:** The numeric revisionId you want to fetch.

The response object holds the following attributes:

- **mindmapId:** The mind map id.
- **revisionId:** The id of the revision.
- **userId:** The id of the user who created this revision.
- **timestamp:** The UNIX timestamp of the creation date of this revision.
- **root:** The actual mind map data of this revision.

revisionRestore

URL: /api/oauth2/v1/revisionRestore
HTTP method: GET
Scope: write
Parameters:

- mindmapId: UUID
- revisionId: NUMBER

Returns:

```
true
```

Explanation:
This call is related to mindmapSave, but instead of providing the full JSON for the mind map, you specifically request to restore and older existing revision. Therefore the whole overwriteToken logic isn't needed here. Of course this call also only works for creators and editors of the mind map. The parameters therefore simply are:

- **mindmapId:** The UUID of the mind map you want to restore a revision for.
- **revisionId:** The numeric revisionId you want to restore.

If the call succeeds it will simply return true. Like mindmapSave this will invalidate all existing sessions and set true refresh flag to "rev_restore".

Collaboration management methods

collaborationList

URL: /api/oauth2/v1/collaborationList
HTTP method: GET
Scope: read
Parameters:

- mindmapId: UUID

Returns:

```
{
	collaborators: [
		{
			userId: UUID,
			name: STRING,
			role: ROLECODE
		},
		...
	],
	pending: [
		{
			invitationId: UUID,
			email: STRING,
			role: ROLECODE,
			url: STRING
		},
		...
	]
}
```

Explanation:
This call fetches all the information about the collaboration set up of a mind map. The parameters are:

- **mindmapId:** The UUID of the mind map you want to query the info about. You must have access to this mind map (collaboration role creator, editor or viewer)

As can be read on http://mind42.com/guide/editor/collaboration Mind42 differentiates between publishing and collaborating. While publishing just allows users to view a mind map, collaboration allows to add other users to the mind map to edit it. The mind map does not have to be public to use collaboration.
When setting up a collaboration, the owner of a mind map invites other users. This happens via email. The user who received the invitation must have (or create) a Mind42 account. When accepting the invitation (by clicking the link received in the invitation email), the collaboration mind map will be added to the invited users account with the role specified by the owner: editor (allowed to make changes) or viewer (just allowed to view changes in real time). The viewer role therefore allows the read only sharing of a mind map without being public (publishing).
The collaborationList command returns all info about the current collaboration state, and therefore includes all registered collaborators, all outstanding invitations, and the owner of the mind map.

Here are the details about the return values:

- **collaborators:** An array of collaboration users, containing there userId, display name and collaboration role. This is basically the same as the collaborators property of mindmapGet.
- **pending:** An array of sent out invitations that haven't been accepted yet. Contains the invitationId, the email the invitation was

sent to, the rolecode and the URL for the receiver to accept the invitation. This URL can be shown to the owner of the mind map after creating the invitation in case the actual invitation mail gets lost, and the owner wants to send the invitation himself to the invited recipient.

collaborationInvite

URL: /api/oauth2/v1/collaborationInvite
HTTP method: POST
Scope: write
Parameters:

- mindmapId: UUID
- emails: STRING
- text: STRING
- role: ROLECODE

Returns:

```
true
```

Explanation:
This call creates and sends collaboration invitations. The parameters are:

- **mindmapId:** The UUID of the mind map you want to create an invitations for. You must be the creator of this mind map, otherwise the call will fail
- **emails:** The email address or a list of email addresses the invitation is being sent to. If it's a list of email addresses, they have to be separated either using a "," or ";" or newline. The call might fail if one of the email addresses is clearly wrong (regex check) or no MX record for the email host of the email address can be found.
- **text:** An additional invitation message from the user which will be added to the invitation email.
- **role:** The role code for the new collaborations. This can only be "ed" for editor or "vi" for viewer. It's not possible to add additional creators.

If the call succeeds pending invitations will be created (which can be seen in "pending" of collaborationList) and invitation emails will be sent out. As mentioned in the parameter description, this call might not only fail for insufficient rights for the mind map, but also if there is a problem sending the emails, or with the email addresses. If everything worked without any errors the call will simply return true.

collaborationRemove

URL: /api/oauth2/v1/collaborationRemove
HTTP method: GET
Scope: write

Parameters:

- mindmapId: UUID
- userId: UUID (OPTIONAL)

Returns:

```
true
```

Explanation:
This call serves a double use. A creator of a mind map (role “cr”) can use it to remove other collaborators of a mind map (with the roles “ed” and “vi”) by supplying a userId. Collaborators on the other hand can use it only to remove themselves from a collaboration (no userId given). A mind map where the user only has “ed” or “vi” access to, also shows up in his or her mind map list. But the user of course can’t call “mindmapDelete” to get rid of it, since only the creator of the mind map can delete it. Therefore the user can only remove him- or herself from the collaboration, therefore “giving up” the right to access the mind map. The parameters are:

- **mindmapId:** The UUID of the mind map where you want to remove a collaboration.
- **userId:** Supplied only if used by the creator of a mind map to remove a specific user. The creator can’t remove his or her own userId. All other collaborators can only remove their own collaboration and must not supply a userId.

If the call succeeds the collaboration will be removed and the call simply returns true.

collaborationRemoveInvitation

URL: /api/oauth2/v1/collaborationRemoveInvitation
HTTP method: GET
Scope: write
Parameters:

- mindmapId: UUID
- invitationId: UUID

Returns:

```
true
```

Explanation:
This call deletes an invitation that has not been accepted yet (pending). This allows creators to revoke invitations that have been sent out accidentally or aren’t needed any longer. The parameters are:

- **mindmapId:** The UUID of the mind map where you want to remove a collaboration invitation for.

- **invitationId:** The invitationId to remove.

If the call succeeds the collaboration invitation will be removed and the call simply returns true.

Publishing methods

publishedGet

URL: /api/oauth2/v1/publishedGet
HTTP method: GET
Scope: read
Parameters:

- mindmapId: UUID

Returns:

```
{
	title: STRING,
	published: BOOLEAN,
	publishedPublic: BOOLEAN,
	description: STRING,
	tags:[
		STRING,
		...
	],
	urls: {
		public: STRING,
		embed: STRING
	],
	role: ROLECODE
}
```

Explanation:
This call retrieves all the information about the publishing state of a mind map. The parameters are:

- **mindmapId:** The UUID of the mind map you want to query the info about. You must have access to this mind map (collaboration role creator, editor or viewer)

As can be read on http://mind42.com/guide/editor/collaboration Mind42 differentiates between publishing and collaborating. When publishing a mind map you basically make a publicly available read only version of the mind map available to everybody who has the link to the mind map. The public map page will show the mind map, a description and tags entered by the user, and the possibility to like and comment the map. So unlike a “viewer” collaborator visitors of published maps will not see the mind map editor. Furthermore users have the decision, if they publish their mind map, whether they also want to include it in the public mind map gallery on Mind42.com, so that maybe other people find their map (through popular, search or related matches) and comment their work. This is called publishing it publicly. It’s also possible to embed the mind map (e.g. in blogs or websites), which is a separate URL, which only shows the mind map viewer, but no comments, …

Here are the details about the return values:

- **title:** The title of the mind map – in case you need to show it in the UI. The title currently can’t be set by the user, but is taken from the text of the root node.
- **published:** Boolean whether the mind map is published or not.
- **publishedPublic:** Boolean whether the mind map is published publicly or not.
- **description:** The description of the mind map as shown on the public map page (this is plain text, not HTML or so).
- **tags:** An array containing the tags assigned to the mind map. They are used in the public mind map gallery. They can’t contain spaces and commas and are lowercase.
- **links:** An object containing the two different public link types. The public link points to the public map page (with comments, …), while the embed link only shows the map is meant to be used for embedding the mind map e.g. using iframes. This links of course only work when published is true.
- **role:** The collaboration role the requesting user has for this mind map. Only creators can change these values, but all other collaborators can of course receive the public links and so on.

publishedSet

URL: /api/oauth2/v1/publishedSet
HTTP method: POST
Scope: write
Parameters:

- mindmapId: UUID
- published: BOOLEAN
- publishedPublic: BOOLEAN
- description: STRING
- tags: ARRAY

Returns:

```
true
```

Explanation:
This call sets the values regarding mind map publishing. This can only be called by the creator of a mind map and will fail otherwise. The meaning of the parameters is described in publishedGet. The parameters are:

- **mindmapId:** The UUID of the mind map you want to update the publishing settings of (only works for creators).
- **published:** Boolean whether the mind map is published or not.
- **publishedPublic:** Boolean whether a published mind map should appear in the Mind42.com mind map gallery (meaningless if published = false).

- **description:** A description about the mind map entered by the used in plain text (no HTML).
- **tags:** A JSON array containing each tag as a string. The strings may not contain spaces and commas (“ “ and “,”).

The return value (the value of the “data” attribute in the response) is a simple boolean “true”, like for mindmapDelete – if the call doesn’t fail.

Recurring JSON structures

The JSON explained in the API methods often referred to recurring standardized JSON structures. Following are the specifications to those structures.

A word on how reliable these specification are:
Mind42 started in 2007 and has built up a huge amount of old data in its database. The currently described JSON format is the 3rd version. If a revision is requested, that still contains older version JSON it will be converted on the fly. But, it can always happen that a specific mind map is not 100% correct JSON. Something specified as a NUMBER, could be a STRING (or vice versa), some value that should be there, isn’t and so on. Mind42 itself is built to deal as gracefully as possible with this. Missing values are covered by default values our application assigns, wrong data formats are dealed with by parsing strings to numbers if needed and so on. You application should do this as well. At the same time, with the start of this API we won’t accept invalid JSON. The server will either throw an error, or silently use default values or converted values to be stored in the database (as explained above).

RoleCode

Type: STRING
Value: “cr”, “ed” or “vi”
Explanation:
In Mind42 there are three different roles of user participation in a collaborative mind map. There is always one creator (code: "cr") who created the mind map, is able to add and remove collaborators, and to the delete the mind map. Editors (code: "ed") are collaborating users who are allowed to make edits to the mind map. Finally viewers (code: "vi") are allowed to open the mind map, but can't make any changes to it.

Timestamp

Type: NUMBER
Value: UNIX timestamp in milliseconds - e.g. 1371047914248
Explanation:
Most timestamps in Mind42 are represented using UNIX timestamps in milliseconds (milliseconds since 01/01/1970 UTC). Unfortunately currently they are not really based on the UTC, but on the local Austrian time (CET/CEST) which is 1/2 hours off.

UUID

Type: STRING
Value: e.g. "f8f99f37-8866-4f0c-a2f4-dc19acba2b8f"

Explanation:
Mind42 uses standard 128bit UUIDs as described in the Wikipedia:
http://en.wikipedia.org/wiki/Universally_unique_identifier

Mind map root JSON

Type: OBJECT
Value:

```
{
	id: UUID+,
	children: [
		MINDMAPNODEJSON,
		...
	],
	attributes: {
		type: "rootnode"
		text: STRING,
	}
}
```

Explanation:
This is the JSON for a root node, but you can clearly recognize the basic structure of every node in Mind42 here. Every node has an “id”, an array of “children”, and an “attributes” object. The “attributes” object always has a “type” attribute. In the case of the root node this type is set to “rootnode”. Here a general explanation of the object attributes:

- **id:** The node id. Basically it’s just a string that has to be unique within the mind map. Newer Mind42 versions use UUIDs for this, but this also could be a longer or shorter string. (Therefore it says “UUID+” as the type)
- **children:** An array of child nodes. See the various Mindmap JSON node types for details.
- **attributes:** An object containing the nodes attributes:
    - **type:** “rootnode” for root nodes
    - **text:** The node caption of the node. This string is HTML formatted (newlines represented using <br>, < using &lt; and so on)

Mind map node JSON

Type: OBJECT
Value:

```
{
      id: UUID+,
      children: [
            MINDMAPNODEJSON,
            ...
      ],
      attributes: {
            type: "container"/"image",
            text: STRING,
            font: FONTJSON,
            icon: ICONCODE,
            links: LINKJSON,
            note: STRING,
            todo: [
                  TODOJSON,
                  ...
            ],
            image: null/IMAGEJSON,
            lastEditor: null/UUID,
            lastEdit: null/TIMESTAMP
      }
}
```

Explanation:
This is the JSON for a node. The basic structure, as explained in the root node, applies. Here's the explanation about the attributes:

- **type:** "container" or "image". This decides whether an attached image is rendered embedded in the mind map (layout), or if it's a text node that maybe also has an image attached, but not shown in the layout.
- **text:** The node caption of the node. This string is HTML formatted (newlines represented using <br>, < using &lt; and so on)
- **font:** Explained in "Mind map font JSON"
- **icon:** Explained in "Mindmap icon code"
- **links:** Explained in "Mindmap link JSON"
- **note:** Stores the note of a node. This is HTML encoded text. Currently the Mind42 editor only offers formatting options for headings, unordered list, ordered lists and bold text, but basically all HTML is shown.
- **todo:** Array of todos, as explained in "Mindmap todo JSON"
- **image:** null, if not image is referenced, or an "Mindmap image JSON" object. If type is "container", the image should not be shown in the mind map, but it should be pointed to like a link. If it's "image", the image should be shown within the mind map layout using the image JSON width and height values.
- **lastEditor:** A user UUID which stores the last user which changed the node. This is currently not used.
- **lastEdit:** A timestamp which stores the last time the node was changed. This is currently not used.

Mind map font JSON
Type: OBJECT

Value:

```
{
	color: STRING,
	size: STRING,
	bold: STRING/BOOLEAN,
	italic: STRING/BOOLEAN,
	underlined: STRING/BOOLEAN
}
```

Explanation:
This object stores the formatting options for a node:

- **color:** In Mind42 the selected color is currently only applied on the lines, not on the font. This line color is inherited to the children, unless otherwise specified. The default color is "#8971c1". The colors are usually rendered with 60% alpha, so they seem a little bit brighter in the app. The default value is "inherit" – this makes the previously explained inheritance mechanism kick in. Mind42 supports 10 colors. These are the supported colors:
    - #fc6e6e
    - #fea852
    - #8ac25b
    - #28cca3
    - #3fbaee
    - #6589cd
    - #8971c1
    - #bd6cc6
    - #e96398
    - #777777
- **size:** Mind42 offers slight variations in the rendered font size of the nodes. There are three size variations: "small", "medium", "large", which are usually rendered with 11px, 13px and 15px. The default value (which defaults to medium rendering) is "default". So overall, the supported values are:
    - default
    - small
    - medium
    - large
- **bold:** Might be "default", in which case Mind42 renders 1st level nodes bold, higher level nodes regular. If it's not "default", it has to be a boolean true or false.
- **italic:** Might be "default", in which case Mind42 renders the node non italic. If it's not "default", it has to be a boolean true or false.
- **italic:** Might be "default", in which case Mind42 renders the node non underlined. If it's not "default", it has to be a boolean true or false.

Mind map icon code

Type: STRING
Value: e.g. "", "star" or "star;shield"

Explanation:
The icon code is a string that stores none, one or more icons that are shown with a node. If no icon is shown, it’s an empty string. If one icon is shown, it’s simply the icon name. If more than one icon is shown, the icon names are concatenated using “;”. These are the supported icon names:

- star
- shield
- award
- thumb_down
- thumb_up
- lock
- key
- accept
- add
- comments
- telephone
- email
- book
- photo
- lightbulb
- lightning
- help
- information
- warning
- clock
- bell
- bug
- emoticon_smile
- emoticon_unhappy
- heart
- user
- cart
- coins
- dollar
- euro
- flag_red
- flag_green
- flag_blue
- flag_yellow
- flag_pink
- nr1
- nr2
- nr3
- nr4
- nr5

- nr6
- nr7
- nr8
- nr9
- nr10
- progress_0
- progress_25
- progress_50
- progress_75
- progress_100

Mind map link JSON

Type: OBJECT
Value:

```
{
	url: STRING, (OPTIONAL)
	wiki: STRING, (OPTIONAL)
	mail: STRING, (OPTIONAL)
	map: UUID (OPTIONAL)
}
```

Explanation:
Mind42 supports four different link types: Web links, Wikipedia links, Mail links and Mind map links. Each node can have one of each type assigned, which is represented using a JSON object with the link type as key, and the linked resource as value. If a node has no link, this is an empty object ({}). The link types:

- **url:** A simple STRING containing the URL of a website (including the protocol). E.g. http://mind42.com
- **wiki:** A simple STRING containing the URL to an English Wikipedia page (including the protocol). Currently only links to the English Wikipedia are supported. E.g. http://en.wikipedia.org/wiki/Tea
- **mail:** A simple STRING of an email address.
- **map:** A UUID referencing another mind map on Mind42. While this theoretically could be anything, this is only useful if the user has actually access to the linked mind map. This is not always the case. For example if the creator of a mind map linked another map only he or she has access to, but a collaborator doesn't.

Mind map todo JSON

Type: OBJECT
Value:

```
{
	progress: STRING,
	priority: STRING,
	date: TIMESTAMP,
	description: STRING
}
```

Explanation:
This objects represents one todo in the todo list of a node. It has the following properties:

- **progress:** A string representing the progress in percent. Only the values “0”, “25”, “50”, “75” and “100” are supported.
- **priority:** A string representing the priority of the todo. “0” for low, “1” for normal, and “2” for high.
- **date:** A timestamp representing the due date of the todo. Currently only the date part of this timestamp is used.
- **description:** A text description of the todo. This string is plain text – not HTML like the node caption.

Mind map image JSON

Type: OBJECT
Value:

```
{
	src: STRING,
	width: STRING,
	height: STRING
}
```

Explanation:
This objects represents an image attached to a node. It has the following properties:

- **src:** A public accessible URL pointing to an image on the WEB.
- **width:** A string saving the width in pixels (without unit) the image should be shown with (e.g. smaller than the real image size). This can be a floating point number (e.g. “76.3”).
- **height:** A string saving the height in pixels (without unit) the image should be shown with (e.g. smaller than the real image size). This can be a floating point number (e.g. “76.3”).

Mind map delta JSON

Type: ARRAY
Value:

```
[
	MINDMAPDELTACREATEJSON,
	MINDMAPDELTAUPDATEJSON,
	MINDMAPDELTADELETEJSON,
	MINDMAPDELTAMOVEJSON,
	...
]
```

Explanation:
This array lists deltas (changes) that have to be applied to the mind maps last known state. You send changes you made to the server using the sessionDelta command, and will receive changes you don’t know about from the server as response. These deltas have to be sent and applied in the order they are listed in the array. There are four types of deltas: CreateDeltas, UpdateDeltas, DeleteDeltas and MoveDeltas. See the according JSON documentation sections for details.

Mind map delta CREATE JSON

Type: OBJECT

Value:

```
{
      userId: STRING (SERVER ONLY),
      action: ”create”,
      id: STRING,
      parentId: STRING,
      index: NUMBER,
      attributes: MINDMAPNODEJSON.attributes (OBJECT)
}
```

Explanation:
This objects represents a new node that should be inserted into the mind map:

- **userId:** The userId this change originated from (e.g. to visualize live changes with different colors per user). This is only set when receiving deltas from the server. It doesn’t need to be set when sending changes from the client to the server.
- **action:** Create deltas always have the action “create”
- **id:** As described for the MindMapNodeJson the client has to create an ID String (e.g. UUID) for the node. The id doesn’t matter, as long as it’s unique within the mind map. If it already exists, sending such a delta will fail.
- **parentId:** The node ID of the parent node this new node should be inserted into. E.g. another nodes id, or the root node id. If this node id doesn’t exist on the servers version of the mind map sending this delta will fail.
- **index:** A numeric index at which position of the parent nodes children the new node should be inserted. It’s a zero based index. 0 means that it should be the first child of the parent, or that it should be inserted before the first existing child of the parent, 1 that it should be the second node (meaning that it should be inserted after the first existing child, or before the second existing child), and so on.
- **attributes:** All the attributes of the new node as described in MindmapNodeJson.

Mind map delta UPDATE JSON

Type: OBJECT
Value:

```
{
      userId: STRING (SERVER ONLY),
      action: ”update”,
      id: STRING,
      attributes: MINDMAPNODEJSON.attributes (PARTIAL OBJECT)
}
```

Explanation:
This objects represents changes to an existing node:

- **userId:** The userId this change originated from (e.g. to visualize live changes with different colors per user). This is only set when receiving deltas from the server. It doesn’t need to be set when sending changes from the client to the server.

- **action:** Update deltas always have the action “update”
- **id:** The ID of the node to update.
- **attributes:** You only have to send the attributes which actually changed. Besides that the attribute specification as described in MindmapNodeJson apply.

Mind map delta DELETE JSON

Type: OBJECT
Value:

```
{
	userId: STRING (SERVER ONLY),
	action: ”delete”,
	id: STRING
}
```

Explanation:
This objects represents the removal of a mind map node:

- **userId:** The userId this change originated from (e.g. to visualize live changes with different colors per user). This is only set when receiving deltas from the server. It doesn’t need to be set when sending changes from the client to the server.
- **action:** Delete deltas always have the action “delete”
- **id:** The ID of the node to remove from the mind map.

Mind map delta MOVE JSON

Type: OBJECT
Value:

```
{
	userId: STRING (SERVER ONLY),
	action: ”move”,
	id: STRING,
	parentId: STRING,
	index: NUMBER
}
```

Explanation:
This objects represents the move of a mind map node to a new parent:

- **userId:** The userId this change originated from (e.g. to visualize live changes with different colors per user). This is only set when receiving deltas from the server. It doesn’t need to be set when sending changes from the client to the server.
- **action:** Move deltas always have the action “move”
- **id:** The ID of the node to move.
- **parentId:** The node ID of the new parent node.
- **index:** The numeric index (as described in the MindmapDeltaCreateJson) at which position the node should be inserted at the new parent.