ELO Sync

Examples

Table of contents

Log on a desktop application	3
Create a synchronization job	5
Update synchronization job	10
Delete synchronization job	11
Run synchronization job	12
Enable/disable synchronization job	13
Resolve synchronization conflict	14
Edit approval	16
Read synchronization log	18
Use of SignalR	19

Log on a desktop application

The following C# code is an example showing how the access token for ELO Sync can be identified for a .NET Core desktop application:

```
static async Task<string> SignInUserAndGetTokenUsingMSAL(string[] scopes)
{
   string authority = $"<instance>/<tenant-id>"; // e.g. https://login.microsoftonline.com/

   // Initialization of the MSAL library
   IPublicClientApplication application = PublicClientApplicationBuilder.Create("<own-clien
                                          .WithAuthority(authority)
                                          .WithDefaultRedirectUri()
                                          .Build();


   AuthenticationResult result;
   try
   {
      // If authentication has already been performed, the session will be reused.
      var accounts = await application.GetAccountsAsync();
      result = await application.AcquireTokenSilent(scopes, accounts.FirstOrDefault())
         .ExecuteAsync();
   }
   catch (MsalUiRequiredException ex)
   {
      // If no authentication has been performed yet, a browser window will open where the
      result = await application.AcquireTokenInteractive(scopes)
         .WithClaims(ex.Claims)
         .ExecuteAsync();
   }


   return result.AccessToken;
}

string[] scopes = new string[] { "<elosync-client-id>/.default" };

string bearerToken = await SignInUserAndGetTokenUsingMSAL(scopes);
```

After this, authentication returns an access token for the application that has to be used for every request to ELO Sync (sent in the HTTP header `Authorization` as `Bearer`).

For more information, see Microsoft identity platform code samples for authentication and authorization

Create a synchronization job

Examples showing how different synchronization job types can be created are shown here. At least the name of the synchronization job (name), synchronization direction (syncDirection), and the synchronization target (syncTargetSystem) must be specified.

You can enter any value for the name, and one of the following values is permitted for the synchronization direction:

syncDirection:

- "ToElo"
- "ToThirdSystem"
- "TwoWay"

The following values are currently supported as the synchronization target:

syncTargetSystem:

- "Sharepoint"
- "OneDrive"

Publishing job

The following example shows how to create a publishing job where documents and folders are transferred from ELO to SharePoint. In the example request, all the mandatory fields required for execution are set.

Other optional fields can also be entered.

You will find an overview of the structure of the synchronization job object with all the fields here: SyncJobEntity

If values are missing that are required for execution, such as the synchronization source and target paths, they can also be added/entered later via update.

This is explained in more detail in the example Update a synchronization job.

> **Information**
>
> For Publishing jobs, no information is required in the metadata form, as the third-party systems handle metadata differently. If you want to apply the metadata anyway, this can be specified via metadata mapping.

```
POST /odata/syncjobs?repository=repo1
Content-Type: application/json
Header: Authorization: <bearer_token>
```

```
{
    "name":"Publish ELO -> Sharepoint",
    "syncDirection": "ToElo",
    "syncTargetSystem": "Sharepoint",
    "eloSystemSyncTarget":{
        "folderName":"<elo-folder-name>",
        "folderId":"<elo-folder-id-int>"
    },
    "sharepointSyncTarget":{
        "listId":"<sharepoint-list-id>",
        "siteId":"<sharepoint-site-id>",
        "name":"<sharepoint-target-display-name>"
    }
}
```

Archiving job

In the following example, an archiving job that archives the contents from a SharePoint Online folder to an ELO folder is created. In the example request, all the mandatory fields required for execution are set.

Other optional fields can also be entered.

You will find an overview of the structure of the synchronization job object with all the fields here: SyncJobEntity

```
POST /odata/syncjobs?repository=repo1
Content-Type: application/json
Header: Authorization: <bearer_token>

{
    "name":"New SyncJob",
    "syncDirection": "ToElo",
    "syncTargetSystem": "Sharepoint",
    "masks":[
        {
            "eloMaskId":"<mask-id-int>",
            "name":"<mask-display-name>",
            "type":"DocMask"
        },
        {
            "eloMaskId":"<mask-id-int>",
            "name":"<mask-display-name>",
            "type":"FolderMask"
        }
    ],
```

```
    "eloSystemSyncTarget":{
        "folderName":"<elo-folder-name>",
        "folderId":"<elo-folder-id-int>"
    },
    "sharepointSyncTarget":{
        "listId":"<sharepoint-list-id>",
        "siteId":"<sharepoint-site-id>",
        "name":"<sharepoint-target-display-name>"
    }
}
```

Synchronization job

The following example shows how a synchronization job can be created directly with all the mandatory fields required for execution.

Other optional fields can also be entered.

You will find an overview of the structure of the synchronization job object with all the fields here: SyncJobEntity

```
POST /odata/syncjobs?repository=repo1
Content-Type: application/json
Header: Authorization: <bearer_token>

{
    "name":"Sync ELO <-> Sharepoint",
    "syncDirection": "TwoWay",
    "syncTargetSystem": "Sharepoint",
    "masks":[
        {
            "eloMaskId":"<mask-id-int>",
            "name":"<mask-display-name>",
            "type":"DocMask"
        },
        {
            "eloMaskId":"<mask-id-int>",
            "name":"<mask-display-name>",
            "type":"FolderMask"
        }
    ],
    "eloSystemSyncTarget":{
        "folderName":"<elo-folder-name>",
        "folderId":"<elo-folder-id-int>"
    },
    "sharepointSyncTarget":{
```

```
        "listId":"<sharepoint-list-id>",
        "siteId":"<sharepoint-site-id>",
        "name":"<sharepoint-target-display-name>"
    }
}
```

Job with all properties

Different properties can be entered depending on the job type. Properties that do not exist for the respective job type are ignored.

The following example shows which properties can be specified in general for a synchronization job:

```
POST /odata/syncjobs?repository=repo1
Content-Type: application/json
Header: Authorization: <bearer_token>

{
    "name":"New SyncJob",
    "syncDirection": "ToElo",
    "syncTargetSystem": "Sharepoint",
    "isActive":true,
    "masks":[
        {
            "eloMaskId":"<mask-id-int>",
            "name":"<mask-display-name>",
            "type":"DocMask"
        },
        {
            "eloMaskId":"<mask-id-int>",
            "name":"<mask-display-name>",
            "type":"FolderMask"
        }
    ],
    "eloSystemSyncTarget":{
        "folderName":"<elo-folder-name>",
        "folderId":"<elo-folder-id-int>"
    },
    "settings":[
        {
            "category":"Approvals",
            "name":"MaxSyncDocumentSize",
            "value":"5242880", // Example: 5MB in Byte
            "hierarchyContext":"SyncJob"
        },
```

```
        {
            "category":"Approvals",
            "name":"MaxSyncableDocuments",
            "value":"50", // Example: 50 Documents
            "hierarchyContext":"SyncJob"
        },
        {
            "name":"ShMetadataMappings",
            "category":"MetadataMappings",
            "value":"{\"eloField\":\"Ordner\\\\ELOINDEX\",\"thirdSystemField\":\"3\\\\Share
            "hierarchyContext":"SyncJob"
        },
        {
            "name":"ShOnlineMetadataExportType",
            "value":"AsJson",
            "hierarchyContext":"SyncJob"
        },
        {
            "name": "MaxFolderReferencesDepthPublish",
            "value": "50",
            "hierarchyContext": "SyncJob"
        },
        {
            "name": "ShOnlineArchivedEntriesReplacementMode",
            "value": "ReplaceByUrlFile",
            "hierarchyContext": "SyncJob"
        },
        {
            "name": "SyncConflictResolutionStrategy",
            "value": "1",
            "hierarchyContext": "SyncJob"
        }
    ],
    "sharepointSyncTarget":{
        "listId":"<sharepoint-list-id>",
        "siteId":"<sharepoint-site-id>",
        "name":"<sharepoint-target-display-name>"
    },
    "timeTrigger":{
        "timePeriod":"Weekly",
        "timeZoneId":"Europe/Berlin",
        "cronJobFormatValue":"* 00 22 * * 1",
        "isActive":true
    }
}
```

Update synchronization job

To update a synchronization job, all you have to do is specify the properties that have changed. The properties that are not specified are then kept without changes.

> **Please note**
>
> The target system (`syncTargetSystem`) and the synchronization direction (`syncDirection`) cannot be changed once a job has been created!
>
> If you want to change the target system or the job type/synchronization direction, you have to create a new job and delete the existing one.

When requesting to change a synchronization job, the ID of the job must be specified both in the path and in the body under `syncJobId`.

The following example shows how the name of a synchronization job is changed:

```
PUT /odata/syncjobs(<syncjobid>)?repository=repo1
Content-Type: application/json
Header: Authorization: <bearer_token>

{
    "name":"New Syncjobname",
    "syncJobId":<syncjobid>
}
```

Delete synchronization job

A synchronization job can be deleted by calling the following endpoint with the synchronization job ID:

```
DELETE  /odata/syncjobs(<syncjobid>)?repository=repo1
Header: Authorization: <bearer_token>
```

Run synchronization job

A synchronization job can be run manually by calling the following endpoint with the synchronization job ID:

```
POST  /odata/syncjobs/run(<syncjobid>)?repository=repo1
Header: Authorization: <bearer_token>
```

The request starts the synchronization job but does not wait for its completion.

SignalR can be used to wait for the completion of a job.

Cancel running synchronization job

A currently running synchronization job can be canceled using the cancel endpoint:

```
POST  /odata/syncjobs/cancel(<syncjobid>)?repository=repo1
Header: Authorization: <bearer_token>
```

Enable/disable synchronization job

A synchronization job can be enabled or disabled by calling one of the corresponding endpoints. This determines whether the synchronization job will be executed automatically based on the configured time trigger.

Enable synchronization job

```
POST /odata/syncjobs/activate(<syncjobid>)?repository=repo1
Header: Authorization: <bearer_token>
```

Disable synchronization job

```
POST /odata/syncjobs/deactivate(<syncjobid>)?repository=repo1
Header: Authorization: <bearer_token>
```

Alternative: Edit synchronization job

Alternatively, the status of the synchronization job can also be set by updating the synchronization job. The property `"isActive"` must be set, and the values `true` or `false` are permitted here.

Resolve synchronization conflict

Resolving a conflict requires the ID and the resolution for the conflict.

The resolution consists of the ID of the element whose changes you want to transfer.

It is not currently possible to transfer a combination of changes from both elements. This must be done manually and the synchronization job run again.

Query synchronization conflicts

To find out whether there are conflicts and, if so, which items are causing conflicts, the conflicts can be queried via the following endpoint:

```
GET  /odata/syncconflicts?repository=repo1
Header: Authorization: <bearer_token>
```

This returns the sync conflicts that have occurred.

> **Information**
>
> The `"firstSystem"` and `"secondSystem"` properties can be used to determine which system the items belong to. The `"firstId"` and the `"secondId"` are the IDs of the conflict items.

Conflict resolution

Once it has been determined which item belongs to which system, the conflicts can be resolved. The corresponding itemId (*firstId* or *secondId* of the conflict) must be specified as the `"Resolution"`.

The example below shows the resolution of two conflicts, one with the ELO version and the other with the SharePoint Online version.

The IDs must match the IDs of the synchronization item, which can be identified from the conflict in question as described above.

```
POST  /odata/syncconflicts/resolve?repository=repo1
Content-Type: application/json
Header: Authorization: <bearer_token>

[
    {
        "SyncConflictId": 1,
        "Resolution": "<elo-itemId>"
    },
    {
```

```
        "SyncConflictId": 4,
        "Resolution": "<sp-itemId>"
    }
]
```

Edit approval

To resolve an approval, the corresponding ID and the new approval status must be specified. The status of the approval must also be specified. The valid values can be found in ApprovalStatus.

Update an approval

An individual approval can be edited, specifying the approval ID, by setting the status:

```
PUT  /odata/approvals(<approval-id>)?repository=repo1
Content-Type: application/json
Header: Authorization: <bearer_token>

{
    "approvalId": "<approval-id>",
    "status": "ApprovedOnce"
}
```

Update multiple approvals at once

Multiple approvals can also be edited at once. The following endpoint can be used:

```
POST  /odata/approvals/batchupdate?repository=repo1
Content-Type: application/json
Header: Authorization: <bearer_token>

[
    {
        "id": "<approval-id-1>",
        "changes": {
            "approvalId": "<approval-id-1>",
            "status": "ApprovedOnce"
        }
    },
    {
        "id": "<approval-id-2>",
        "changes": {
            "approvalId": "<approval-id-2>",
            "status": "Approved"
        }
    },
    {
        "id": "<approval-id-3>",
        "changes": {
```

```
            "approvalId": "<approval-id-3>",
            "status": "Declined"
        }
    }
]
```

Read synchronization log

The synchronization log documents what exactly was synchronized during a job run.

Read log for the last run of a synchronization job

The endpoint below can be used to get the synchronization log from the last run of a synchronization job. This example also includes the synchronization details in the results to give a more exact overview of the synchronization.

```
GET  /odata/synchronizationactivities/latest(syncJobId=<syncjobid>)?repository=repo1&$expan
Header: Authorization: <bearer_token>
```

Use of SignalR

ELO Sync uses SignalR to notify the clients of events such as the execution and processing of synchronization jobs, processing of approvals, or resolution of conflicts. Thanks to the implementation of SignalR, a client application can easily respond to the events triggered by ELO Sync.

You will find an introduction to SignalR at the following link:

What is SignalR?

SignalR events in ELO Sync

ELO Sync provides the following SignalR events:

SyncJobSyncStateChanged:

- Event that notifies about a change in the synchronization job's synchronization status. In this case, for example whether it is currently running, is waiting, is successfully completed, was completed or canceled with conflicts or errors.
- A synchronization job object is included with the properties SyncJobId, OwnerId, EloOwnerId, RepositoryKey, possibly an error message Error, SyncState, IsActive, and SyncTargetSystem.

SyncJobUpdated:

- Event that notifies about changed synchronization jobs.
- A synchronization job object is included with the properties SyncJobId, OwnerId, EloOwnerId, RepositoryKey, possibly an error message Error, SyncState, IsActive, and SyncTargetSystem.

SyncJobAdded:

- Event that notifies about added synchronization jobs.
- A synchronization job object is included with the properties SyncJobId, OwnerId, EloOwnerId, RepositoryKey, possibly an error message Error, SyncState, IsActive, and SyncTargetSystem.

SyncJobRemoved:

- Event that notifies about removed synchronization jobs.
- A synchronization job object is included with the properties SyncJobId, OwnerId, EloOwnerId, RepositoryKey, possibly an error message Error, SyncState, IsActive, and SyncTargetSystem.

SyncJobModeChanged:

- Event that notifies about a change in the synchronization job's status. In this case, whether it is active or inactive.
- A synchronization job object is included with the properties SyncJobId, OwnerId, EloOwnerId, RepositoryKey, possibly an error message Error, SyncState, IsActive, and SyncTargetSystem.

ApprovalStatusChanged:

-

Event that notifies about a change in the approval status.

- An approval object with the properties from the ApprovalEntity is provided.

ConflictResolved:

- Event that notifies about the resolution of a conflict.
- A SyncConflictEntity object for the corresponding resolved conflict is included,

SyncJobFinished:

- Event that notifies about the end of a synchronization job.
- A synchronization job object is included with the properties SyncJobId, OwnerId, EloOwnerId, RepositoryKey, possibly an error message Error, SyncState, IsActive, and SyncTargetSystem.