ELO Sync

Synchronization

Table of contents

Synchronization

Concepts

This section describes the concepts used for synchronizing data in ELO Sync.

Pipeline

A two-stage pipeline is currently used for synchronization, where multiple middleware components are registered.

Each stage of the pipeline must be completed before the next stage is started. This ensures that different changes in multiple systems cannot interfere with each other, thereby reducing the complexity required to process these changes.

A major disadvantage of this approach is that all changes are stored in the memory until the next stage can be processed, leading to higher memory consumption.

Generally, synchronization is done top down, meaning folders are processed before their contents, ensuring that any changes to folder contents can trust that their parent folders have already been synchronized completely.

The first stage includes the analysis and implementation of changes unrelated to deletion. The data of all synchronized systems is retrieved and compared with the metadata stored in ELO Sync to determine which changes have been made since the last synchronization. All recognized changes are converted into modifications for later implementation.

If deletions are found, these are first scheduled normally like other changes and then transferred to the second stage of the pipeline. This is necessary to prevent entries that were not deleted from being deleted. Example: A user moves a file from folder A to folder B and then deletes folder A. If deletion is not scheduled for a later time and it is not guaranteed that the parents are synchronized before their children, deletion would delete folder A, including the moved file.

For more information, refer to the Pipeline chapter.

System

A *system* describes software, a server, or a part thereof that contains data to be synchronized.

This can be a SaaS system such as OneDrive or SharePoint Online, or a local ELO repository.

System Provider

A software component that provides the implementation for access to a system. It is only used sometimes when delegation of system creation is required.

Collection

Collections are a subset of the contents of a system that provide abstract access to these contents.

Some systems use several collections (especially when contents cannot be uniquely identified), or just a single collection.

The exact details depend on the data provided by the system.

Subcollection

The term *subcollection* is used for collections that are contained in another collection, or that are referenced by an element in a collection.

Synchronization Entry

A *synchronization entry*, or simply *entry*, is an abstract unit that describes a thing that is synchronized between several systems.

An entry does not contain its own data, but is instead the superset of all items that are synchronized together.

If it is necessary to uniquely identify an entry, we recommend using the identity of the associated data mapping. This identity is maintained as long as the individual entry items exist.

Synchronization Item

A *synchronization item* describes a certain item in a synchronized system, such as a file or a folder. ELO Sync can also synchronize more abstract data such as e-mails or chats.

Data Mapping

The *data mapping* contains the metadata of a synchronized entry. This generally means that the data mapping contains hash values or similar elements to also be able to recognize changes at a later time.

The data in the mapping can be stored for a long time and therefore must not contain potentially protected information (such as PII). This is another reason for the use of hash algorithms (or comparable methods) for storing all data in the mapping.

Structure Mapping

The *structure mapping* of an entry stores its relative position in the synchronized structure.

In contrast to data mapping, structure mapping does not hash or encrypt data, as it must be reconstructable during later synchronization runs.

Field

The data of the synchronized elements is accessed through *fields*. Each field enables access to a single, non-divisible piece of information on the element.

Non-divisible here means that this information should either be synchronized completely or not at all. It does not mean that information cannot be divided and combined.

System Field

The *system fields* are an exception that must be provided by all synchronized systems to enable synchronization.

The system fields currently defined are: *Name*, *Content*, and *Parent*.

Of these fields, only the *Name* field must have contents.

The two *Content* and *Parent* fields are required but can be left blank.

Field Mappings

By using *field mappings*, individual fields are compared across different systems so that the synchronization process knows which data must be compared and exchanged.

Normally, the user configures the field mappings when creating a synchronization job.

The system fields are excepted from the mappings configured by the user. These fields are always mapped automatically.

In ELO, it is possible to create multiple documents with different metadata forms in the same synchronized folder. This means the field mappings are not simply 1:1 mappings between two systems.

The fields in the synchronization process are always considered n:m relationships, but when certain elements are synchronized, the mapping rules should result in a 1:1 mapping.

Example

The mappings defined for the synchronization between an ELO repository and OneDrive are listed here as an example.

```
flowchart LR

subgraph system-a[ELO]
    subgraph mail[Mask: E-Mail]
        field-a2[Field: Subject]
    end
    subgraph rechnung[Mask: Bill]
        field-a3[Field: Title]
    end
    subgraph document[Mask: *]
        field-a1[Short name]
    end
end

subgraph system-b[OneDrive]
    field-b1[Filename]
end

field-a1 <--> field-b1
field-a2 <--> field-b1
field-a3 <--> field-b1
```

The file in OneDrive only has the file name, but there are multiple metadata forms available in the ELO repository with which this file name can be synchronized.

However, if a certain document is synchronized with the corresponding file in OneDrive, the metadata form of the document is known and the mappings are therefore reduced to a single mapping.

Information

This is just an example.

In
a production scenario, the file name is a system field and is always synchronized with the *Short name* system field in ELO.

If the document has the metadata form *Bill* in ELO, the following synchronization would be performed:

```
flowchart LR

subgraph system-a[ELO]
    subgraph invoice["Document (Bill)"]
        field-a3[Field: Title]
    end
end

subgraph system-b[OneDrive]
    subgraph file[File]
        field-b1[File_name]
    end
end
```

```
field-a3 <--> field-b1
```

Configuration

Synchronization job configuration settings via the user interface

These configuration settings can be configured globally or for each synchronization job via the user interface.

Metadata mapping

To include the metadata from the columns in the SharePoint libraries and the ELO fields in the synchronization, they must be connected to one another. This is done via metadata mapping. You will find more information here.

Metadata

This chapter describes how the synchronization of metadata from or to SharePoint Online works and can be configured.

What is metadata?

In SharePoint Online, metadata refers to the non-system-relevant columns in document libraries. You can find more information on SharePoint Online, for example, on creating document libraries or user-defined columns, in the SharePoint documentation.

In ELO, metadata is the fields of index and aspect forms. You will find more information on ELO index forms and aspect forms in the documentation.

Supported columns

In SharePoint Online, only specific columns can be transferred to ELO.

User-defined columns are supported if they are one of the following types:

- Text
- Date and time
- Multi-line text
- Number
- Yes/No
- Hyperlink
- Currency

Of the system columns that SharePoint provides, the following columns are supported:

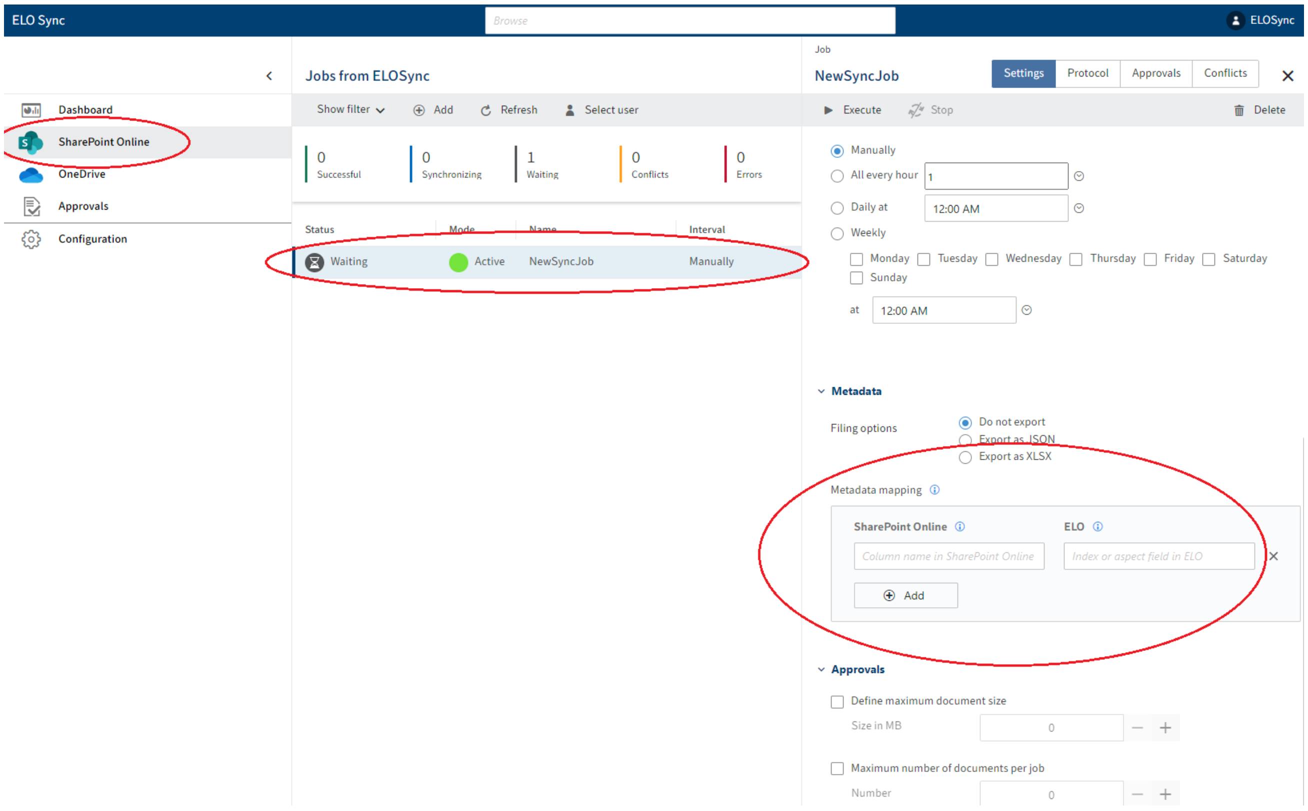
Name	Metadata rule
Compliance object ID	`*\*\ComplianceAssetId`
Title	`*\*\Title`
Created	`*\*\Created`
Changed	`*\*\Modified`
Check-in comment	`*\*\_CheckinComment`
Item child count	`*\*\ItemChildCount`
Folder child count	`*\*\FolderChildCount`
Name setting	`*\*\_ComplianceFlags`
Storage name	`*\*\_ComplianceTag`
Storage name used.	`*\*\_ComplianceTagWrittenTime`
Name used by	`*\*\_ComplianceTagUserId`
Comment count	`*\*\_CommentCount`
Like count	`*\*\_LikeCount`

Name	Metadata rule
App created by	`*\*\AppAuthor`
App edited by	`*\*\AppEditor`

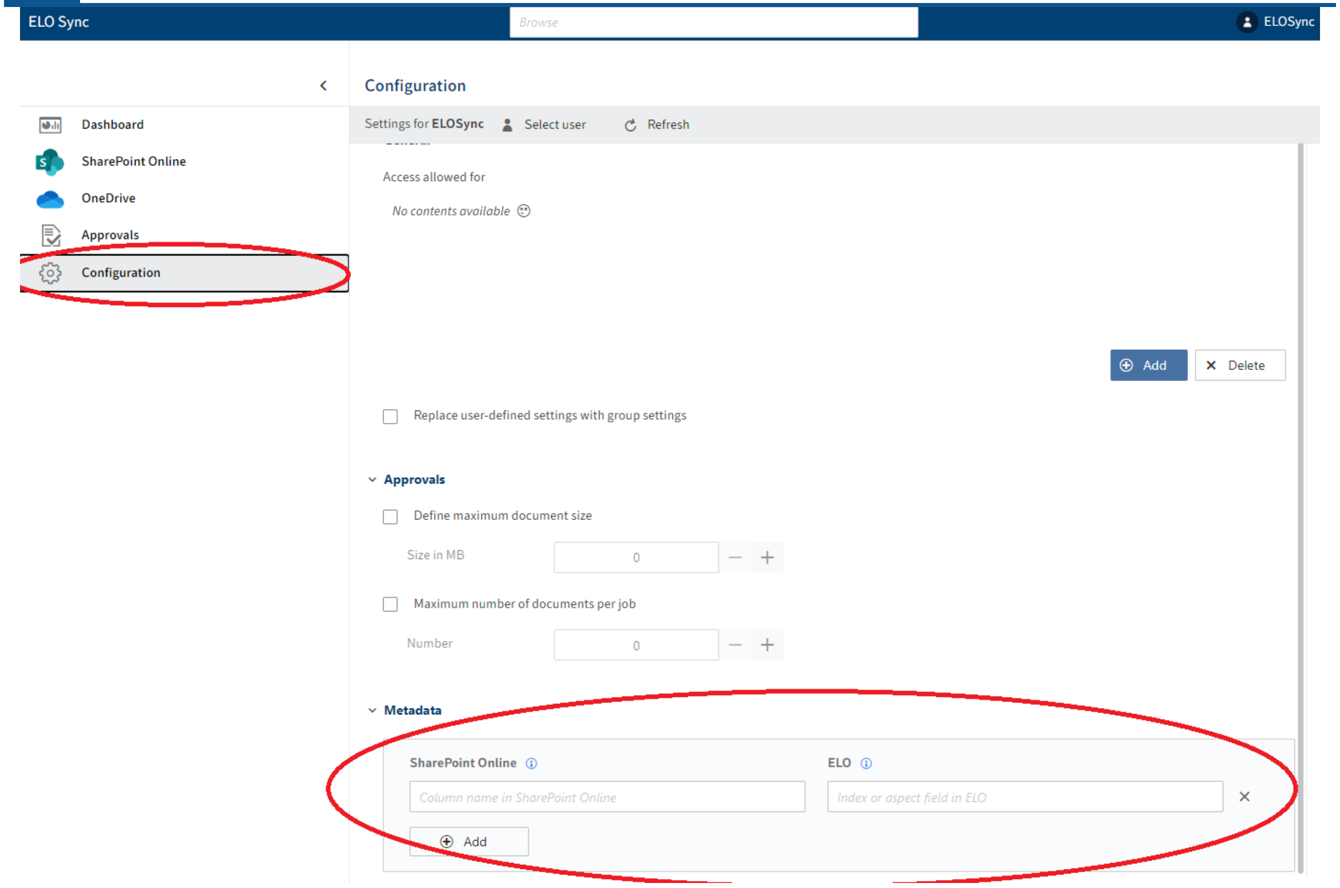
These columns are read-only, i.e. you cannot transfer metadata from ELO to SharePoint.

Metadata mapping

To link metadata fields between SharePoint Online and ELO, a field mapping must be defined. This can either be created for each synchronization job in the respective configuration, or globally in the settings, whereby the synchronization job configuration links have priority.



Metadata mappings in the synchronization job configuration



Metadata mappings in the global configuration

If a field could not be mapped, the metadata is not transferred for the field.

Metadata mapping structure

Mappings can consist of several parts and are separated by a "".

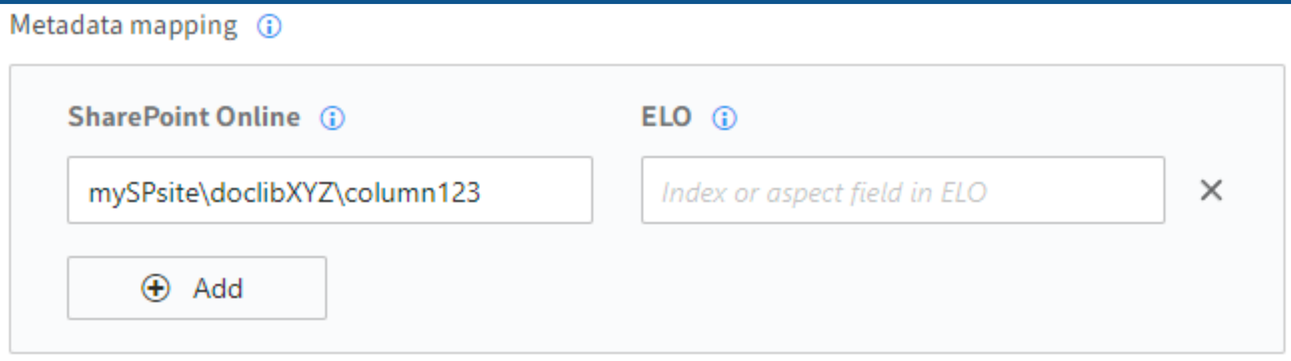
Mapping structure for SharePoint Online

The structure of a SharePoint Online mapping may look as follows:

SiteName\ListName\ColumnName

Site and list names are optional here. To prevent confusion (e.g. a column from another list), mappings should contain as much information as possible.

Example:



mySPsite\doclibXYZ\column123

Index or aspect field in ELO

Mapping structure for ELO index and aspect fields

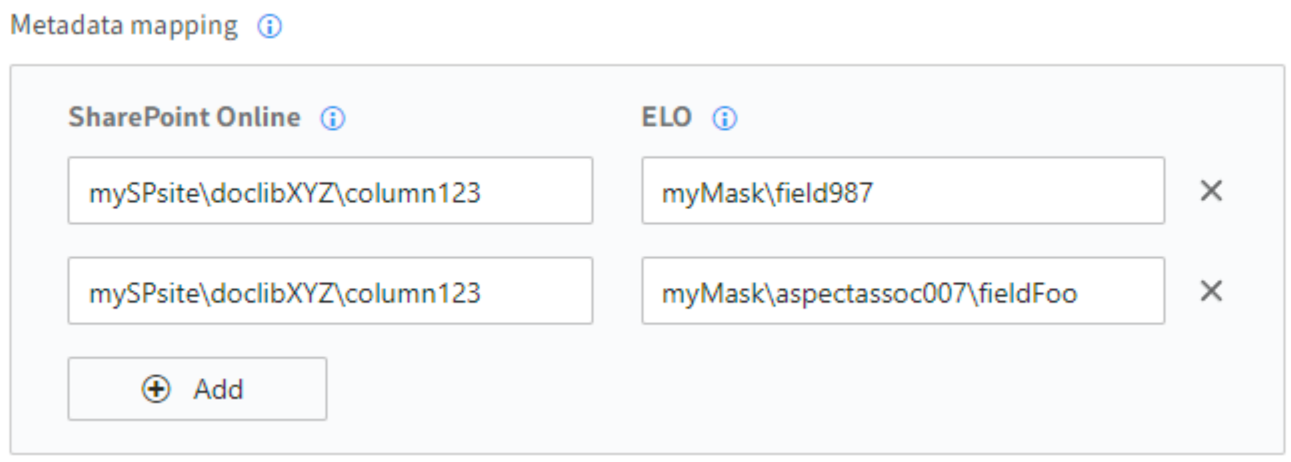
The structure of an ELO mapping may look as follows:

Index fields: *MaskName\FieldName*

Aspect forms: *MaskName\AspectAssociation\FieldName*

Form names and aspect mappings are optional here. To prevent confusion (e.g. a column from another list), mappings should contain as much information as possible.

Example:



SharePoint Online	ELO
mySPsite\doclibXYZ\column123	myMask\field987
mySPsite\doclibXYZ\column123	myMask\aspectassoc007\fieldFoo

Wildcards

In addition, the two wildcards "*" and "?" can be used in the mappings. Here, "?" represents any character and "*" represents an unlimited number of any characters.

Example:

mySPsite\doclib*\column1?3 myMask*\fieldF??

Field mapping

Field mappings can be made both in the settings of the respective synchronization jobs and in the global settings. The mapping in the synchronization job settings has a higher priority here, i.e. if a field mapping was defined in the synchronization job settings and in the global settings, only the synchronization job settings are taken into account.

The respective mappings are resolved from the smallest unit to the largest, i.e. from back to front. Therefore, a mapping is first performed on the column or field name.

Based on the example from the section Mapping structure for SharePoint Online `mySPSite\\doclibXYZ\\column123`, the mapping is resolved in the following order:

1. *column123*
2. *doclibXYZ*
3. *mySPSite*

Supported target URLs for SharePoint

- URLs created via the share function in SharePoint, usually by right-clicking a file or folder and then selecting *Copy link*
- The browser URLS for most SharePoint sites

Detailed list of supported URLs

This is a list of SharePoint URLs that are supported, where wildcards (*/?) are used for the variable parts.

The patterns are tested from the top down. If a pattern matches, it is used and the remaining patterns are skipped over.

URL Pattern	Information
https://tenant.sharepoint.com/:*:/s/*	Sharing link; selects the library/list if it refers to a file
https://tenant.sharepoint.com/⭕/r/sites/*	OneNote notebook; selects the containing library
https://tenant.sharepoint.com/sites//.aspx	URL to the form; the direct target may not be selected, but rather a parent target
https://tenant.sharepoint.com/sites//*Lists/*	URL to the list view; the direct target may not be selected, but rather a parent target
https://tenant.sharepoint.com/sites//_*layouts/*	URL to the library view; the direct target may not be selected, but rather a parent target
https://tenant.sharepoint.com/sites//_/*	URL to the site API; selects the corresponding site
https://tenant.sharepoint.com/sites/*	Other URLs; the direct target may not be selected, but rather a parent target
https://tenant.sharepoint.com/	Selects the root site
https://tenant.sharepoint.com/_/	SharePoint API; selects the root site

There are restrictions when the root site is selected, as the API provided by Microsoft does not provide any contents for the root site itself.

It is possible to enter a URL for the root site, but it is not possible to select an actual SharePoint synchronization target.

This can change in the future, if Microsoft changes its API, but is not currently supported.

Synchronization

Default pipeline for synchronization

This document describes the default pipeline ELO Sync uses to synchronize data.

Depending on the type and direction of the synchronization job, elements are added to or removed from the pipeline.

Overview

A synchronization pipeline can consist of an unlimited number of stages; the current default pipeline uses two stages.

Each stage contains a series of middleware components that each perform an individual step for a synchronization entry.

The next stage only begins once all entries in the previous stage have been processed. This ensures that some operations are always performed after others.

```
flowchart LR

start(Quelle) --> def-pipeline-stage

subgraph def-pipeline [default pipeline]
    direction TB
    subgraph def-pipeline-stage [default stage]
        direction TB
        automapping(automatic mapping)
        --> itemresolution(handle linked elements)
        --> deduplication(resolve duplicate elements)
        --> entry-limit(note count limit)
        --> size-limit(note size limit)
        --> classification(classify elements)
        --> difference(determine differences between elements)
        --> conflicts(resolve conflicts)
        --> execution(apply non-destructive changes)
    end
    subgraph deletion-pipeline-stage [deletion stage]
        execution2(apply destructive changes)
    end

    def-pipeline-stage --> deletion-pipeline-stage
end
```

The default and delete phases are performed sequentially rather than in parallel.

This ensures that changes are always performed in the correct order and destructive changes do not prevent the successful execution of other changes.

In addition, this makes it easier to use non-destructive changes, as they do not have to be checked for any additional restrictions.

Reasons for the two-stage pipeline

There is currently only one reason for using a two-stage pipeline:

The combination of moving an element and deleting its (indirect) parent element between two synchronizations.

In all systems currently supported, deleting a folder also deletes its entire contents. This leads to a problem if an element is moved from a folder and the folder is then deleted.

For performance reasons, the synchronization code is always in parallel for each entry, so it is possible that the folder is deleted in the target system before the element has been moved.

Information

It seems an obvious solution could be to map the recorded changes in the order they occur, but this can lead to minor errors in borderline cases:

1. It is not guaranteed that the synchronized systems use precise clocks.
2. Synchronization may take place well after the changes, and with bidirectional synchronization, the object is moved in one system and the folder deleted in the other.

As the synchronization code cannot rely on the precision of the reported timestamp, another solution must be found so that these operations can be performed successfully.

The current solution is to record all the destructive changes in a queue, which is executed once all the other changes have been processed.

Example

Assume the following initial structure in both systems:

- Folder *A*
    - File *F1*
    - File *F2*
- Folder *B*

Between two synchronization runs, a user moves the file *F1* to *B* and then deletes folder *A*.

To repeat this operation correctly, the changes have to be performed in the same order.

Please note that move and delete actions do not have to be performed on the same system and therefore it cannot be guaranteed that the timestamp (or similar) can be compared.

Information

Timestamps can differ between systems, for example, because a system simply has the wrong time set, or an NTP time server was temporarily unavailable.

There are lots of reasons for unreliable timestamps.

A general recommendation for reliable systems is not to use timestamps when deciding on actions.

Source

The source for the synchronization retrieves the elements from the individual systems for processing and makes them available to other components.

All readable systems are inserted in the processing queue for the source, and then a generator block is created that calls the elements and provides them for further processing. The generator block starts a number of parallel workers between 1 and the number of systems to be synchronized.

Each worker retrieves an element from the processing queue and performs different actions, depending on the element type.

For a system, the worker calls its root collection and inserts all the root elements made available by this collection in the processing queue. The elements are grouped in blocks of up to eight so that subsequent processing can combine multiple retrievals or changes.

If the processed element is a provider element from another collection, this collection is called and its root elements are inserted in the processing queue. Otherwise, the children of the entry are called and divided into groups of eight entries and then processed like entries in a system.

Executor

The executor executes each configured pipeline stage serially, and all entries within one stage in parallel.

For the first stage, the source block is taken from the source and processed; later stages process the queue for their stage as the source block.

After receiving an input context for processing, the executor checks whether the current context has a predecessor and, if so, waits until this is completed. Afterwards, the executor starts the actual execution of the middleware components configured for the stage.

Middlewares

Automatic mapping

This automatically maps objects between different systems based on different criteria.

The default configuration for this middleware maps elements based on their relative path within the synchronized structure.

Example

If a document library in SharePoint Online contains a file under *Project/Marketing/Presentation.pptx*, a search is performed in the ELO repository for *Project*, for a *Marketing* folder within the folder, and in this folder a document with the short name *Presentation* with a current version that has the file extension *.pptx*.

Handle linked elements

This is required if previous steps in the pipeline could only determine IDs of elements, but did not retrieve the elements themselves.

This middleware then searches for the metadata of these elements in the database and, if possible, queries the associated systems these elements belong to as to whether the elements exist (or were deleted).

This ensures that all the subsequent middleware components have at least access to the metadata and the status of these elements. If elements have been deleted, these elements are replaced with corresponding markers that can be examined by other middleware.

Resolve duplicate elements

This middleware is used for optimization to prevent elements from being synchronized multiple times in the same synchronization.

> **Information**
>
> The worst-case scenario is using delta mode (only retrieving differences). The connected systems then list each element multiple times for every change.
>
> If 100 changes have been made to an article since the last synchronization, for example, the system could list this article 100 times.
>
> This would lead to at least 100 synchronizations of this element (99 of which would be unnecessary).

Observe count limit

This middleware forces the configured entry limit for a synchronization job.

If no limit is configured, this middleware does nothing.

If the configured limit is reached, an approval is generated automatically, or, if a hard limit is configured, all the other entries are skipped.

No additional elements are synchronized until approval has been confirmed by a user.

> **Information**
>
> All entries are retrieved from the system, even if the limit has already been reached.
>
> This is required to specify the exact number of entries.

- Approval handling

Observe size limit

This middleware forces the configured file size limit for synchronized files.

If no limit is configured, this middleware does nothing.

An approval is generated for each file that exceeds the configured limit, or, if a hard limit is configured, the file is not synchronized.

The file is not synchronized until approval has been granted by a user.

- Approval handling

Classify elements

This middleware ensures that the status classification of objects is correct, or determines their status if unknown.

As with handling linked elements, this middleware is used to ensure that later middlewares have a consistent status for all the synchronized items.

This middleware has the following tasks:

- To ensure that the new objects are marked as new
- To ensure that changed elements are actually marked as changed
- To unsure that unchanged elements are actually marked as unchanged

Determine differences between elements

This middleware performs most of the analysis during synchronization.

It determines which changes were made to elements and whether these changes conflict with other changes.

To begin, a prototype is determined from all the connected elements.

This prototype is used as a comparison template for all other elements.

It then retrieves all fields from the prototype and determines the associated fields for each field.

Each field entry is then checked for changes and, if necessary and possible, their contents are compared.

If there are no conflicting changes, a corresponding object is created that enables the change to be processed later. The modification object contains all the information required to make the change.

If multiple conflicting changes are made in different fields, a conflict is generated. A new conflict is created with all relevant data, including the elements and fields that are affected by this conflict.

Resolve conflicts

This middleware is used to process and resolve conflicts.

If a handler can resolve a conflict, a corresponding modification is created and placed in a queue with the other changes.

If no handler can provide a solution, the conflict is written in the technical log and the synchronization of this entry is canceled to ensure that no incorrect changes are made.

- Refer to Conflict handling for more information.

Apply non-destructive changes

This middleware performs the actual synchronization after all analyses are complete.

The middleware is added to several stages and makes different types of changes, depending on the stage.

The actual implementation of changes is delegated to the registered change handlers, which move changes to later phases, or can even rearrange pending changes in the current phase.

The currently registered handlers behave as follows:

- In the default stage, all changes are performed directly, with the exception of deletions. The execution order depends on the intra entry staging of the changes that were previously defined by the analysis middleware.
- All destructive changes are rescheduled in the deletion stage.
- All remaining scheduled changes are performed in the deletion stage.

Conflict handling

This handler stores conflicts in the database and delivers the resolution if it is stored in the database.

At the end of synchronization, all irrelevant and resolved conflicts are removed from the database as they are no longer needed.

Resolved conflicts are conflicts that have been resolved by a user or a program using the REST API.

Irrelevant conflicts are conflicts that no longer exist. This can be for various reasons, for example, due to the manual deletion of all affected elements.

The current implementation uses a version field in the conflict database table to find irrelevant conflicts. At the start of the synchronization, the version is incremented and new or updated conflicts save the new version number. After synchronization is complete, all conflicts that do not have the new version number are deleted.

Information

There was an alternative approach that used a timestamp to delete all conflicts older than the current timestamp.

For reasons of correctness, this timestamp had to be updated to prevent a conflict that was found during a previous synchronization run and still exists from being deleted.

The problem was that this timestamp can also be shown to users, and it can be confusing if a conflict suddenly has a different timestamp.

Approval handling

Under certain configurable circumstances, approvals may be required for synchronization.

These approvals must be confirmed by a user to enable the synchronization of the relevant items or jobs.

The idea behind this is that an administrator (or users themselves) can set limits to prevent the accidental or excessive synchronization of documents.

The current implementation stores the approvals in the database and provides the configured settings from there.

Information

Approvals are never deleted automatically.

Long-running synchronization jobs can receive many approvals over their lifetime.

The reason for this is that users do not have to keep approving the same actions in future synchronizations.

Known issues and limitations

Incorrect date values

The synchronization code assumes that all date values entered without an explicit time zone are in the UTC time zone.

If there are sources that provide values in the local time zone (or a different one), this can lead to incorrect conversions and therefore incorrect values in other systems.

No unique names in the ELO repository

Currently, all synchronized folders and documents must have unique names in the ELO repository for synchronization.

This restriction exists as most systems that can be synchronized require unique names, whereas this is not the case with ELO.

Currently, only one of the elements with an identical name is synchronized with another system. All other elements are ignored. The element that is synchronized cannot be predicted.

> **Information**
>
> Automatic renaming is planned for a future version.
>
> The exact method for renaming is currently being analyzed.

Special characters in the short name

SharePoint Online and OneDrive have restrictions with regard to the characters that can be used in file and folder names. This restriction does not apply in the ELO repository.

To enable the synchronization of files and folders with these special characters, the short name is checked for problematic characters and these are then encoded with percent-encoding.

The following characters are always encoded: /, \, :, *, ?, ", <, >, |

The encoding of these characters is implemented by the system provider, and not the general synchronization system, as only the system provider is familiar with the character restrictions.

Leading/trailing spaces in the short name

Normally, the ELOix removes leading and trailing spaces in the short name.

As ELO Sync synchronizes with other systems that permit leading and trailing spaces in the file and folder names, all leading or trailing spaces from external systems are encoded with the Unicode character `U+2420`/␠ (symbol for space) and are not removed from the short name.

For more information, refer to the FAQ entry Unusual characters in short name.

Conflicts with unchanged Office documents in SharePoint

After changing the metadata of an Office document in SharePoint Online, the new data is automatically written in the file contents of the associated document.

This results in a new file version that must then be synchronized with ELO.

If a new working version is uploaded to ELO at the same time (for example, by a different user), a conflict is recognized that must be resolved manually.

The metadata within the Office document in SharePoint Online in particular is invisible, and therefore it may appear as if the document has not been changed at all.

References in ELO

References are currently ignored in ELO, as there are unresolved problems in handling changes to associated elements.

Conflicts with automatic mapping

If new elements are found during synchronization and these can be mapped automatically to existing elements, multiple conflicts may result for these elements if their data does not match.

This is a security measure, as ELO Sync does not know the original version and therefore cannot predict what data should be kept and what should be discarded.

Information

Different methods enabling automatic handling of these conflicts are currently in the pipeline. One example: by using the more recent data set.

No duplicate check during filing

If new documents are found that should be filed to ELO, these are always filed as individual documents.

There is no duplicate check and therefore no references are created.

This is in part due to the fact that there are currently unresolved issues with references in ELO and how they are handled in ELO Sync.