ELO XC

Practice

Table of contents

Practice

Log files

ELO XC writes logs at the earliest possible time to reproduce operations at runtime. If log files are not written immediately when the service starts, there are two possible reasons for this:

- The file *ELOxc.xml* is configured incorrectly
- The ELO XC account does not have write permissions on the log directory

For performance reasons, the logs are separated and not stored in a log file.

Settings

You configure the log settings in the *ELOxc.xml* file.

```
<!-- the directory all ELO XC logs are written to -->
<xs:attribute name="LogDir" type="xs:string" use:"optional" default="C:\Temp"/>
<!-- maximum log file age as number of days -->
<xs:attribute name="LogAge" type="xs:int" use="optional" default="0" />
<!-- maximum log detail -->
<xs:attribute name="LogDetail" type="xs:int" use="optional" default="20" />
```

LogDir: *LogDir* is where ELO XC stores the log directory tree. It also contains the Windows service log as well as the ELO XC Manager user interface log.

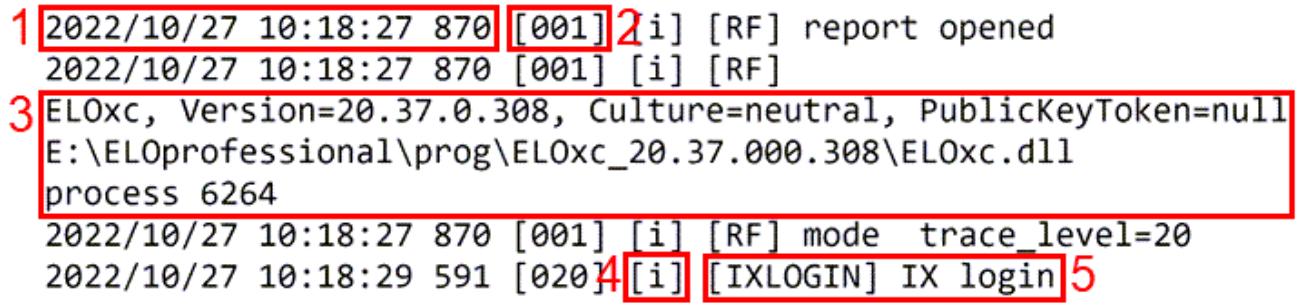
LogAge: You can specify the retention period for logs here. The default value `0` ignores the log age. Other values, i.e. greater than `0`, are interpreted as the maximum log age in days up to a maximum of `365`. Once logs reach the configured maximum age, they are automatically deleted.

LogDetail: The log files in ELO XC do not have settings like *debug* or *info*. Instead, *granular logging* is applied, i.e. the log level (*LogDetail*) accepts values from `0` to `20`. The higher the value, the more detailed the log is. The value `0` means no logs are created.

> **Please note**
>
> If you have enabled automatic deletion of old logs, use a different path for the log directory than the one used for the installation directory. The deletion routine deletes all files in the configured log directory, regardless of whether they are actually logs or not.

Log lines

The log is created line by line at runtime. The time, part of the program, type of message, and a log message are recorded.



1 Timestamp: All timestamps are in universal (time zone independent) time to make it easier to compare data with other programs if you have distributed applications. The three-part structure consists of date, universal time, and milliseconds.

2 Execution path: Logs are assigned to internal execution paths.

3 Program version: When a log is started, information about the executed ELO XC program version is recorded.

4 Marker: Log messages have different markers depending on their type. The following markers can appear:

- [i] – Information is the most common type of log output, and hence the default type.
- [w] – Warnings appear rarely. They are indications that something is not working properly, but are not a reason to terminate the program.
- [e] – You should normally pay attention to regular errors. Most errors need to be fixed to ensure that the program works properly. This message type is always recorded regardless of the logging level.
- [F] – Fatal errors are recorded when there is a risk that processing may fail immediately or has already failed at another point. In such cases, processing is usually aborted immediately. This message type is always recorded regardless of the logging level.
- [x] – External log lines from the Microsoft framework or certain Microsoft components may occasionally appear, provided that the respective assemblies automatically support the ELO XC log.

5 Log message: Log messages consist of a block (or context) and the actual message. The enables you to easily identify the process or part of the program affected, while the message reflects the current processing point.

> **Information**
>
> The standard way to check log files manually is to search for error markers. With [e] or [F], you can quickly find the relevant parts, even in more extensive logs.

Individual log outputs

A directory with the same name is created for each instance. This is where the main log, catalog requests, job directories, and job ZIPs are stored.

The following log outputs are written:

Identifier	Validity	Meaning
Srv	Service	This file contains everything the process does on its own, such as start, basic structure, basic functions, top-level information, UnhandledExceptionCatcher.
Wks	Service	This log contains logs for functions that are triggered interactively (ELO XC Manager).
Main	Instance	This log relates to the instance configuration (e.g. parameter warnings, automatic version updates, license count). It also contains information as to why a configuration may not be viable. If the configuration is valid, the instance starts and logs the job starts.
MainCatalog	Instance	This log contains catalog queries via LDAP or PowerShell. It contains a log of connection issues during creation of a catalog and the list of selected mailboxes.
JobControl	Job	This file contains information about the job controller. It logs the mailbox registrations, item selections, and worker control.
JobCatalog	Job	This file corresponds to *MainCatalog*, but is created by *JobControl* at job start.
Worker	Job	Each worker writes its own log. It logs the processing of the items that are selected by the job controller and transferred to workers.
JobErrors	Job	This log lists all items for which an action tree returns the *not successful* result.
Summary	User	These files contain log summaries created interactively. They are written to the *<LogDir>_Summaries* child folder and can be opened with LogViewer.exe.

> **Please note**
>
> The global Trace option ensures that all http network traffic from the EWS interface is logged in addition to the logs written by ELO XC. This option generates extensive logs and should therefore only be used in exceptions or after consulting product support.

Isolation

Isolating relevant logs is important in a production system since there may be a large number of logs depending on the specified maximum log age. When a problem is reproducible, logs can help you to understand the problem. You have two options to isolate relevant parts of the log and compile important logs:

- Manual compilation
- Automatic compilation

Manual compilation

The manual method is useful if ELO XC Manager is unavailable and you do not have the option to compile logs automatically. You can also use this method if you know where to look for the problem. Proceed as follows:

1. Stop ELO XC.
2. Rename the log directory. This ensures you retain the old logs.
3. Restart ELO XC.
4. Collect the logs until the problem occurs.
5. Close ELO XC.
6. Zip the log directory.

Automatic compilation

You can compile logs automatically with the Log analysis tool. After executing the function, you will find a ZIP file in the *_Summaries* child folder.

Troubleshooting

The following examples are intended to show which log files may be relevant depending on the error scenario:

Error	Log
Settings in ELOxc.xml are incorrect or the service could not be started	Srv
All other logs show no errors, but one occurs	Srv
Start/stop instances	Srv
Configuration interface is not accessible or contains an error	Wks, Srv
Interactive functions (e.g. log analysis, catalog test)	Wks
Validation of the instance configuration fails	Main
The scheduler is not working	Main
License warnings/errors during processing	Main
Version update fails	Main
Catalog connection fails	MainCatalog, JobCatalog
Mailboxes are not found/resolved	MainCatalog, JobCatalog
EWS connection fails	JobControl
Mailbox folders are not processed	JobControl
ELO XC is terminated because a worker is no longer responding	JobControl, Worker
Recipients on journal envelopes/delegates in SharedMailbox are not resolved	Worker, MainCatalog
Documents or metadata are not stored	Worker

Error	Log
Attachments are not found	Worker

Folder determination

Folder determination and selection can be reproduced in the JobControl log. First, the known folders are searched according to the configuration (lines shortened):

```
[EWSFC.ReadAllFolders] included folder types: MSG, REC, ARC, ARCREC


[KFCOL] find known folders ...
[KFCOL] folder group "basic" (required)
[KFCOL]   MsgFolderRoot: id=AQMkADJhOTY1ZmI3LWMyOWQtNDdhMS1hZjQzLTZkYTVmZDNjOWU4


[KFCOL]   Calendar: id=AQMkADJhOTY1ZmI3LWMyOWQtNDdhMS1hZjQzLTZkYTVmZDNjOWU4ZQAuA


[KFCOL]   Contacts: id=AQMkADJhOTY1ZmI3LWMyOWQtNDdhMS1hZjQzLTZkYTVmZDNjOWU4ZQAuA


[KFCOL]   DeletedItems: id=AQMkADJhOTY1ZmI3LWMyOWQtNDdhMS1hZjQzLTZkYTVmZDNjOWU4Z


[KFCOL]   Inbox: id=AQMkADJhOTY1ZmI3LWMyOWQtNDdhMS1hZjQzLTZkYTVmZDNjOWU4ZQAuAAAD


[KFCOL]   Notes: id=AQMkADJhOTY1ZmI3LWMyOWQtNDdhMS1hZjQzLTZkYTVmZDNjOWU4ZQAuAAAD


[KFCOL]   SentItems: id=AQMkADJhOTY1ZmI3LWMyOWQtNDdhMS1hZjQzLTZkYTVmZDNjOWU4ZQAu


[KFCOL]   Tasks: id=AQMkADJhOTY1ZmI3LWMyOWQtNDdhMS1hZjQzLTZkYTVmZDNjOWU4ZQAuAAAD


[KFCOL] folder group "ext" (optional)
[KFCOL]   Drafts: id=AQMkADJhOTY1ZmI3LWMyOWQtNDdhMS1hZjQzLTZkYTVmZDNjOWU4ZQAuAAA


[KFCOL]   Journal: id=AQMkADJhOTY1ZmI3LWMyOWQtNDdhMS1hZjQzLTZkYTVmZDNjOWU4ZQAuAA


[KFCOL]   VoiceMail: id=AAMkADJhOTY1ZmI3LWMyOWQtNDdhMS1hZjQzLTZkYTVmZDNjOWU4ZQAu
```

```
[KFCOL]   JunkEmail: id=AQMkADJhOTY1ZmI3LWMyOWQtNDdhMS1hZjQzLTZkYTVmZDNjOWU4ZQAu

[KFCOL]   SearchFolders: id=AQMkADJhOTY1ZmI3LWMyOWQtNDdhMS1hZjQzLTZkYTVmZDNjOWU4

[KFCOL] folder group "sync" (optional)
[KFCOL]   RecipientCache: id=AQMkADJhOTY1ZmI3LWMyOWQtNDdhMS1hZjQzLTZkYTVmZDNjOWU

[KFCOL]   QuickContacts: id=AQMkADJhOTY1ZmI3LWMyOWQtNDdhMS1hZjQzLTZkYTVmZDNjOWU4

[KFCOL]   ConversationHistory: id=AQMkADJhOTY1ZmI3LWMyOWQtNDdhMS1hZjQzLTZkYTVmZD

[KFCOL]   MyContacts: id=AQMkADJhOTY1ZmI3LWMyOWQtNDdhMS1hZjQzLTZkYTVmZDNjOWU4ZQA

[KFCOL]   IMContactList: id=AQMkADJhOTY1ZmI3LWMyOWQtNDdhMS1hZjQzLTZkYTVmZDNjOWU4

[KFCOL]   PeopleConnect: id=AQMkADJhOTY1ZmI3LWMyOWQtNDdhMS1hZjQzLTZkYTVmZDNjOWU4

[KFCOL]   Favorites: id=AQMkADJhOTY1ZmI3LWMyOWQtNDdhMS1hZjQzLTZkYTVmZDNjOWU4ZQAu

[KFCOL]   AllContacts: id=AQMkADJhOTY1ZmI3LWMyOWQtNDdhMS1hZjQzLTZkYTVmZDNjOWU4ZQ

[KFCOL] folder group "recover" (optional)
[KFCOL]   RecoverableItemsRoot: id=AQMkADJhOTY1ZmI3LWMyOWQtNDdhMS1hZjQzLTZkYTVmZ

[KFCOL]   RecoverableItemsDeletions: id=AQMkADJhOTY1ZmI3LWMyOWQtNDdhMS1hZjQzLTZk

[KFCOL]   RecoverableItemsVersions: id=AQMkADJhOTY1ZmI3LWMyOWQtNDdhMS1hZjQzLTZkY

[KFCOL]   RecoverableItemsPurges: id=AQMkADJhOTY1ZmI3LWMyOWQtNDdhMS1hZjQzLTZkYTV
```

```
[KFCOL] folder group "arc" (optional)
[KFCOL]   ArchiveMsgFolderRoot: id=AQMkAGQxYzJlOWM5LWZjYmYtNDhhYi04OTIwLTFmNDA4Z


[KFCOL]   ArchiveInbox: id=AQMkAGQxYzJlOWM5LWZjYmYtNDhhYi04OTIwLTFmNDA4ZTc5MzQzN


[KFCOL]   ArchiveDeletedItems: id=AQMkAGQxYzJlOWM5LWZjYmYtNDhhYi04OTIwLTFmNDA4ZT


[KFCOL] folder group "arcrecover" (optional)
[KFCOL]   ArchiveRecoverableItemsRoot: id=AQMkAGQxYzJlOWM5LWZjYmYtNDhhYi04OTIwLT


[KFCOL]   ArchiveRecoverableItemsDeletions: id=AQMkAGQxYzJlOWM5LWZjYmYtNDhhYi04O


[KFCOL]   ArchiveRecoverableItemsVersions: id=AQMkAGQxYzJlOWM5LWZjYmYtNDhhYi04OT


[KFCOL]   ArchiveRecoverableItemsPurges: id=AQMkAGQxYzJlOWM5LWZjYmYtNDhhYi04OTIw


[KFCOL] wkfn omitted:   PublicFoldersRoot


[KFCOL] wkfn succeeded: MsgFolderRoot {%MSGROOT}, Calendar {%APPOINTMENT}, Conta


[KFCOL] wkfn done
[COLAMBF-MsgFolderRoot] 38 folders found
[COLAMBF-RecoverableItemsRoot] 4 folders found
[COLAMBF-ArchiveMsgFolderRoot] 4 folders found
[COLAMBF-ArchiveRecoverableItemsRoot] 4 folders found
```

ELO XC first logs the configured folder groups (*folder types*). These are searched and the result of the search is displayed at the end. The available folder variables are now known. Once these are determined, the internal folder structure is created and filters and entry points are applied:

```
[EWSFC.Dump] ########## FOLDERS xc191@xc.local
[EWSFC.Dump] MSG       0-1-1      {%MSGROOT}           ""
[EWSFC.Dump] MSG       0-1-0      {%TASK}              - "Tasks"
[EWSFC.Dump] MSG       0-1-0                           - "Conversation Action Settings"
[EWSFC.Dump] MSG       0-1-0                           - "Files"
[EWSFC.Dump] MSG       0-1-0                           - "Settings for QuickSteps"
```

```
[EWSFC.Dump] MSG       0-1-0      {%DRAFTS}            - "Drafts"
[EWSFC.Dump] MSG       0-1-0                           - "ExternalContacts"
[EWSFC.Dump] MSG       0-1-2      {%DELITEMS}          - "Deleted Items"
[EWSFC.Dump] MSG       0-1-5      {%SENTMAIL}          - "Sent Items"
[EWSFC.Dump] MSG       0-1-0      {%JOURNAL}           - "Journal"
[EWSFC.Dump] MSG       0-1-0      {%JUNK}              - "Junk E-mail"
[EWSFC.Dump] MSG       0-1-1      {%APPOINTMENT}       - "Calendar"
[EWSFC.Dump] MSG       0-1-1      {%CONTACT}           - "Contacts"
[EWSFC.Dump] MSG       0-1-0      {%QUICKCONTACTS}     -- "{06967759-274D-40B2-A3EB-D7F9E73
[EWSFC.Dump] MSG       0-1-0      {%IMCONTACTS}        -- "{A9E2BC46-B3A0-4243-B315-60D9910
[EWSFC.Dump] MSG       0-1-0                           -- "Companies"
[EWSFC.Dump] MSG       0-1-0                           -- "GAL Contacts"
[EWSFC.Dump] MSG       0-1-0                           -- "Organizational Contacts"
[EWSFC.Dump] MSG       0-1-0                           -- "PeopleCentricConversation Buddie
[EWSFC.Dump] MSG       0-1-3      {%RCPTCACHE}         -- "Recipient Cache"
[EWSFC.Dump] MSG       1-1-0      {%NOTES}             - "Notes"
[EWSFC.Dump] MSG       0-1-0                           - "Outbox"
[EWSFC.Dump] MSG    => 0-0-1      {%INBOX}             - "Inbox"
[EWSFC.Dump] MSG       0-1-0                           -- "Sub1"
[EWSFC.Dump] MSG       0-1-0                           - "RSS Feeds"
[EWSFC.Dump] MSG       0-1-0                           - "Sync Issues"
[EWSFC.Dump] MSG       0-1-0                           -- "Conflicts"
[EWSFC.Dump] MSG       0-1-0                           -- "Local Errors"
[EWSFC.Dump] MSG       0-1-0                           -- "Server Errors"
[EWSFC.Dump] MSG       0-1-0      {%CONVHISTORY}       - "Conversation History"
[EWSFC.Dump] MSG       0-1-0                           - "Yammer Root"
[EWSFC.Dump] MSG       0-1-0                           -- "Outgoing"
[EWSFC.Dump] MSG       0-1-0                           -- "Incoming"
[EWSFC.Dump] MSG       0-1-0                           -- "Feeds"
[EWSFC.Dump] MSG       0-1-0                           - "zzzKeep"
[EWSFC.Dump] MSG       0-1-7                           -- "AttTest"
[EWSFC.Dump] MSG       0-1-1000                        -- "TestdataBig"
[EWSFC.Dump] MSG       0-1-1000                        -- "TestdataMedium"
[EWSFC.Dump] MSG       0-1-1000                        -- "TestdataTiny"
[EWSFC.Dump] REC       0-0-0      {%RECITEMSROOT}      "Recover"
[EWSFC.Dump] REC       0-0-0                           - "Calendar Logging"
[EWSFC.Dump] REC       0-0-0      {%RECITEMSDEL}       - "Deletions"
[EWSFC.Dump] REC       0-0-0      {%RECITEMSPURG}      - "Purges"
[EWSFC.Dump] REC       0-0-0      {%RECITEMSVER}       - "Versions"
[EWSFC.Dump] ARC    => 0-0-2      {%ARCMSGROOT}        "Archive"
[EWSFC.Dump] ARC       0-0-0                           - "Files"
[EWSFC.Dump] ARC       0-0-0                           - "ExternalContacts"
[EWSFC.Dump] ARC       0-0-0      {%ARCDELITEMS}       - "Deleted Items"
[EWSFC.Dump] ARC    => 0-0-1                           - "Test"
[EWSFC.Dump] ARCREC    0-1-0      {%ARCRECITEMSROOT}   "ArchiveRecover"
[EWSFC.Dump] ARCREC    0-1-0                           - "Calendar Logging"
```

```
[EWSFC.Dump] ARCREC    0-1-0      {%ARCRECITEMSDEL}    - "Deletions"
[EWSFC.Dump] ARCREC    0-1-0      {%ARCRECITEMSPURG}   - "Purges"
[EWSFC.Dump] ARCREC    0-1-0      {%ARCRECITEMSVER}    - "Versions"
```

The log is formatted as a table at this point. The folder group identifier is at the beginning followed by => for the folders to be included. The three-digit filter status represents the evaluation. The first digit stands for the name filter, and the second one stands for entry points. If one of the two values is 1, the respective folder is filtered, i.e. not processed. The third digit stands for the number of items in total. If a folder doesn't contain any items, it's also filtered.

This is followed by the available folder variable, the level below the respective root, and the display name used after storage in the repository, for example.

Before ELO XC starts processing the folders, the valid folders and roots are output again:

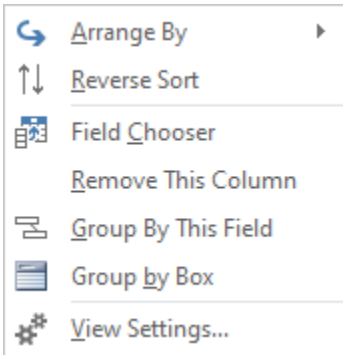
```
[EWSFC.GetValidFolderList] MAILBOX xc191@xc.local VALID FOLDERS: ARC.Archive, MSG.Inbox, AR
```

Show properties

Third-party tools are often used to display and analyze message properties. These tools give you advanced insight into internal and often hidden message data. However, these tools are not available in every situation and environment. Furthermore, the extensive capabilities of the tools are not even required because only property values need to be displayed, and you can use Outlook tools for this.

In this example, you can see how the message class *PidTagMessageClass* and the ELO GUID *EloXcArchivedExt* of an archived message can be displayed.

1. Start Microsoft Outlook.
2. Select the *Preview* view.
3. Right-click to open the context menu.

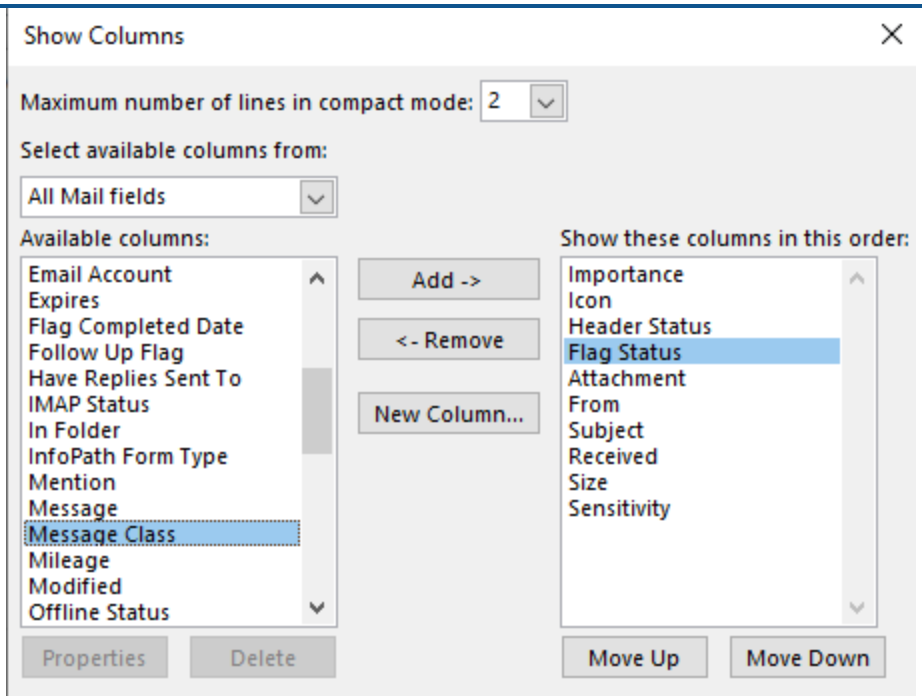


4. Select *View Settings*.

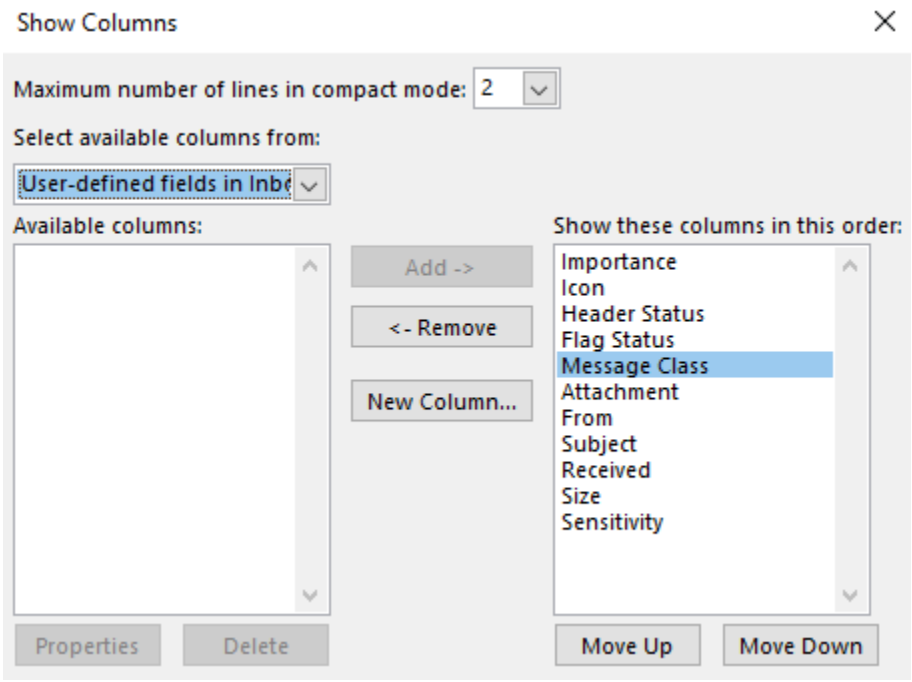
   In the *Advanced View Settings* dialog box, you can edit the columns.

5. Click *Columns...*

   The available columns are grouped differently. The *Frequently used fields* group does not contain the message class. To be able to select it, you need to switch to the *All Mail fields* group.



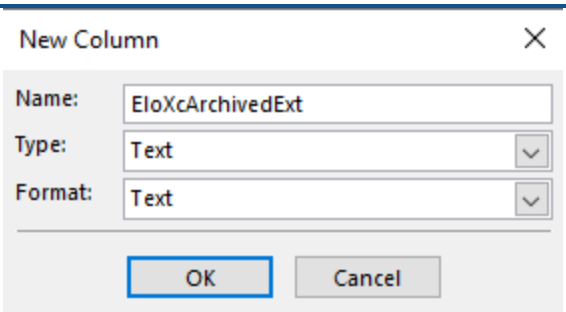
6. Select *Message Class* and click *Add*.



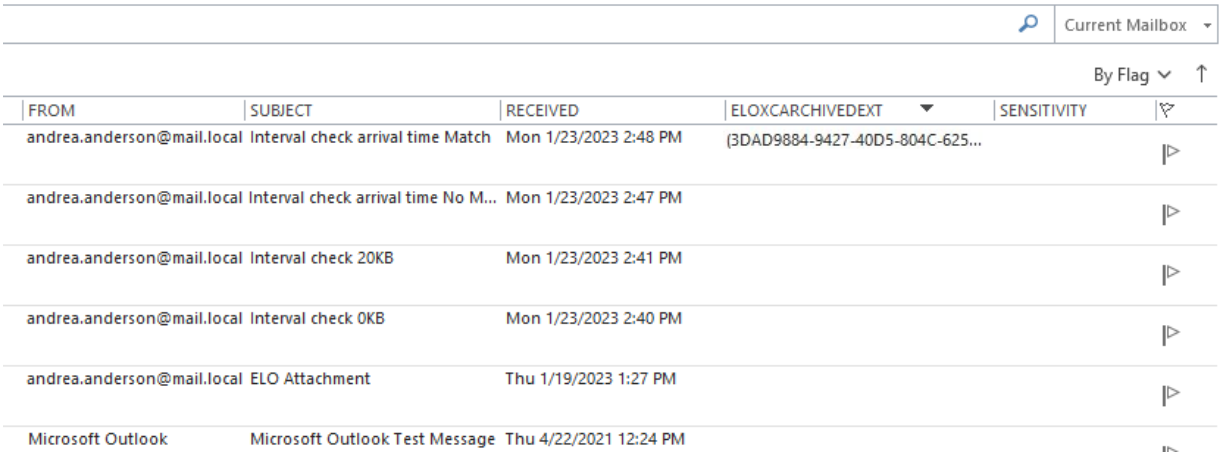
7. To add the ELO XC archiving status, you need to select the *User-defined fields in Inbox* group.

8. Since the property is created by ELO XC and Outlook has no information about it, you now need to create a new column.

9. Click *New Column*.



10. Enter `EloXcArchivedExt` as the name with the type and format `Text` and confirm with *OK*.



Once you have closed the *Advanced View Settings*, you will see *Message class* and *EloXcArchivedExt* in your view.

Very large messages

Message size limits allow you to specify the size of messages a user is allowed to send and receive. When a mailbox is created, there is no default size limit for sending and receiving messages.

In this best practice tip, we show you how to configure size limits in Microsoft Exchange and in the EWS interface.

Exchange

Set the message size limits using the Exchange Management Console.

> **Information**
>
> Refer to this Microsoft documentation when configuring message size limits in Microsoft Exchange.

In the following example, the maximum size of sent messages for *Debra Garcia's* mailbox is set to 25 MB, and to 35 MB for received messages. Enter the cmdlet in the Exchange Management Console PowerShell.

```
Set-Mailbox -Identity "Debra Garcia" -MaxSendSize 25mb -MaxReceiveSize 35mb
```

To verify that the settings were successfully applied, run the following command in the Exchange Management Console PowerShell:

```
Get-Mailbox -Identity <Identity> | Format-List MaxSendSize,MaxReceiveSize
```

EWS

To configure the message size in Exchange Web Services (EWS), run the Microsoft IIS web server.

You need to set the `maxAllowedContentLength` parameter in various IIS web server *web.config* files.

> **Information**
>
> You can find more information in this Microsoft documentation and on the CodeTwo website.

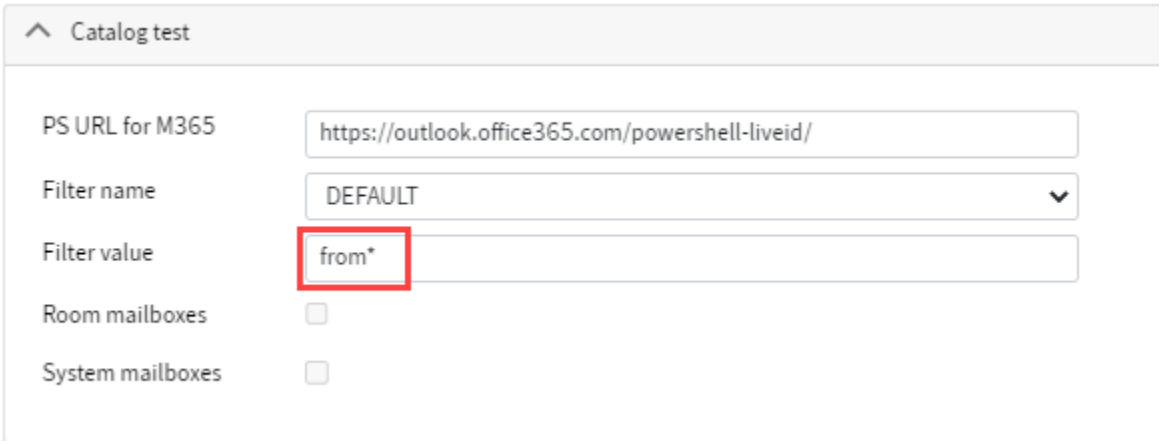
Catalog test with PowerShell

The following section contains some examples of how to use the *Catalog test* tool to apply filters for the *m365* catalog type. You need to be in the instance configuration to do this.

1. Select *Connection test* from the tools menu in the ribbon.
2. Select the *m365* catalog type.
3. Open the *Catalog test* options.

Wildcard

The *Get-Mailbox* cmdlet allows the use of wildcards.

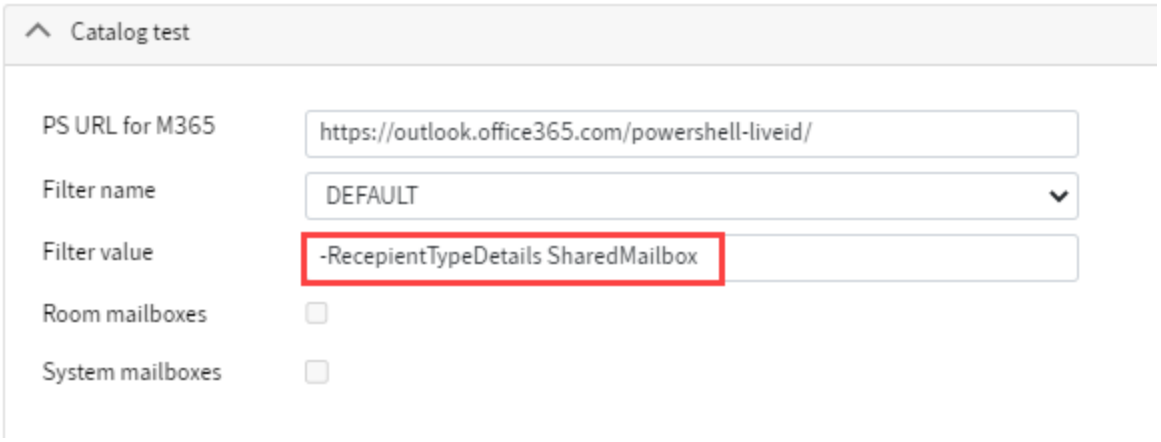


1. Enter *from** as the `filter value`.
2. Click the *Catalog test* button.

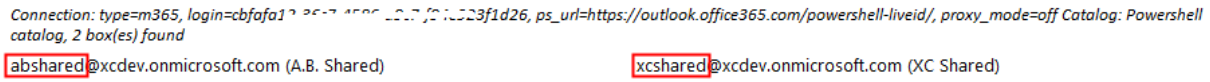
The result of the catalog test is displayed at the bottom. All mailboxes starting with `from` are listed.

SharedMailbox

Here is an example of how to use a *SharedMailbox* filter.



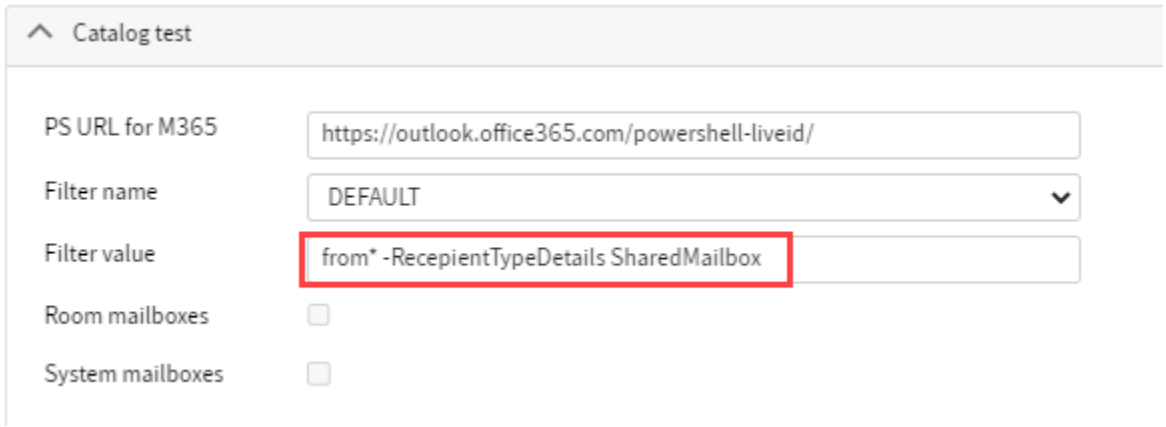
1. Enter `-RecipientTypeDetails SharedMailbox` as the filter value.
2. Click the *Catalog test* button.



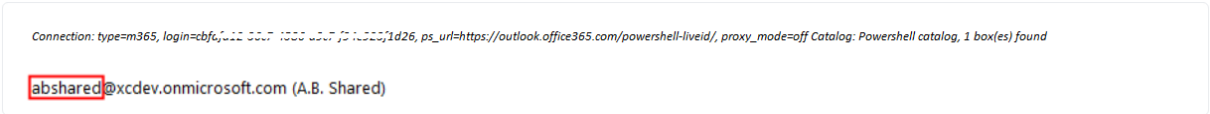
The result of the catalog test is displayed at the bottom. All mailboxes of the *SharedMailbox* type are listed.

Wildcard and SharedMailbox

You can also use a combination of filters.

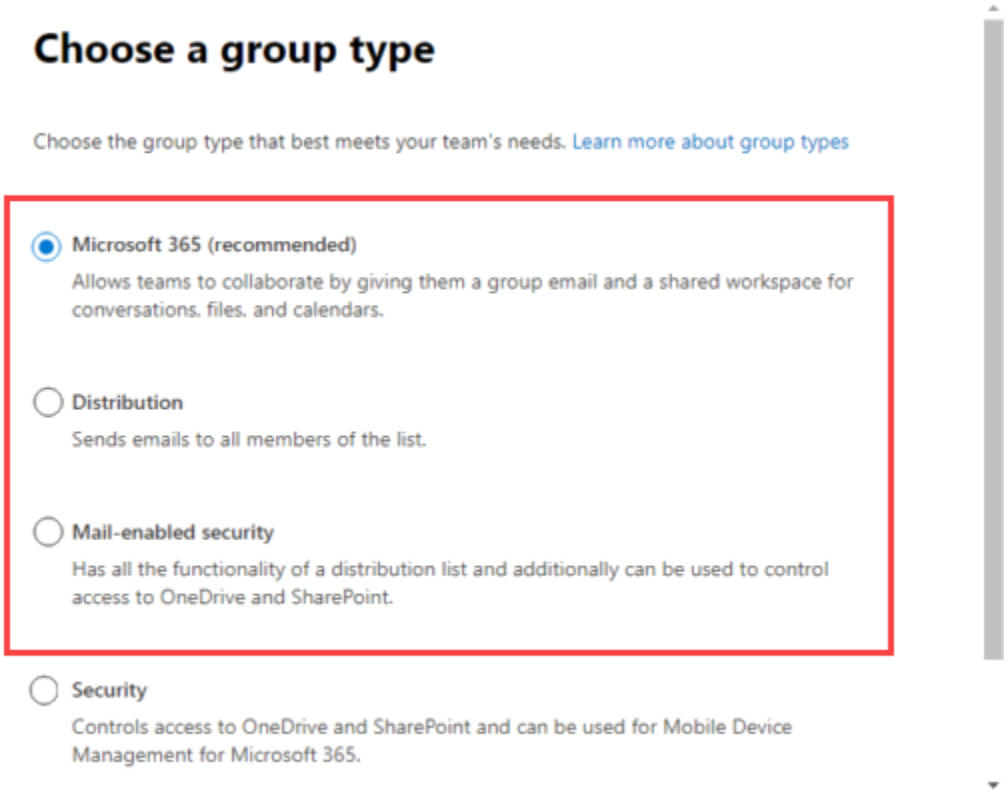


1. Enter `from* -RecipientTypeDetails SharedMailbox` as the filter value.
2. Click the *Catalog test* button.

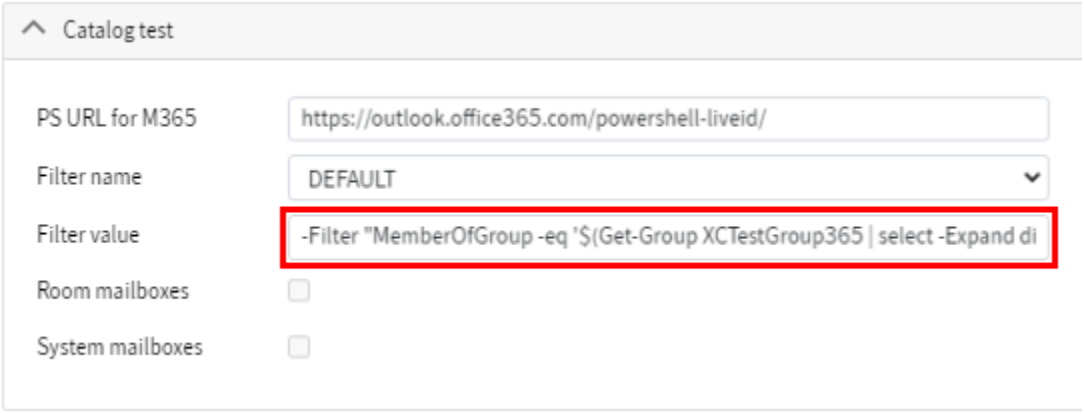


The result of the catalog test is displayed at the bottom. All mailboxes of the *SharedMailbox* type that start with `from` are listed.

Mailbox selection by group



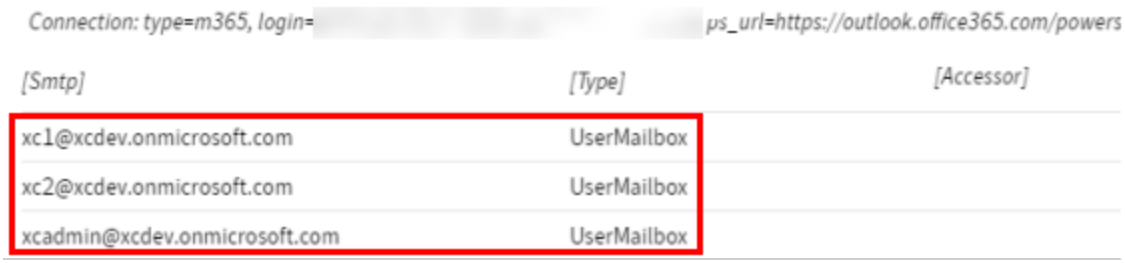
1. In the Microsoft 365 menu, assign the *Microsoft 365*, *Distribution*, or *Mail-enabled security* group types to the mailboxes. The *Security* group type is not supported.
2. Create a group (e.g. with the group name *XCTestGroup365*).
3. Assign mailboxes to this group.



4.

Enter the filter value `-Filter "MemberOfGroup -eq '$(Get-Group 'XCTestGroup365' | select -Expand distinguishedName)'"¶` for example.

5. Click the *Catalog test* button.

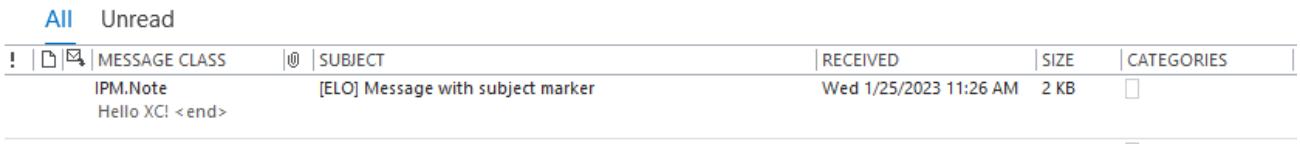


The result of the catalog test is displayed at the bottom. All mailboxes of the group *XCTestGroup365* appear.

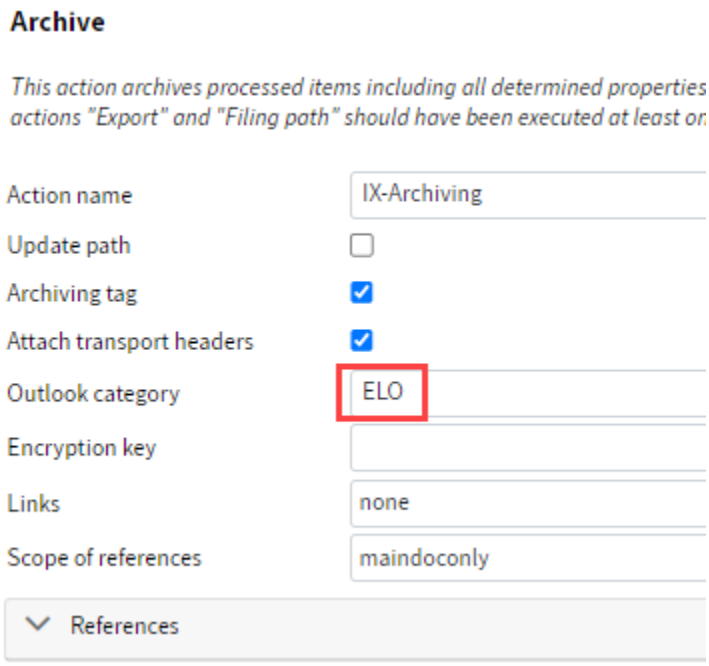
Outlook categories

In this example, we show you how to use Outlook categories in combination with ELO XC to tag different processing states.

Since the icons in Outlook are bound to message classes, ELO XC does without this option to tag processing states. Instead, we recommend setting a tag for the mailbox owners by subject.

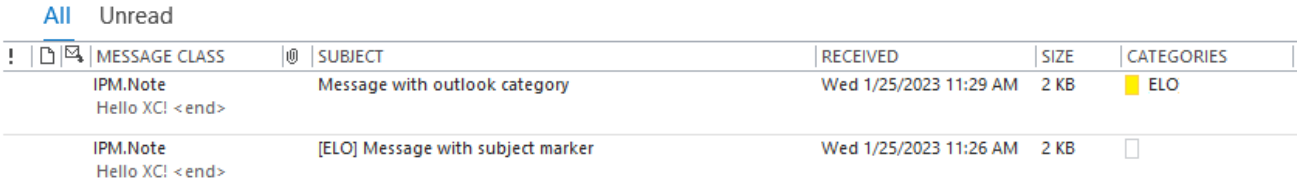


However, in cases where you don't want to use this, it is also possible to tag e-mails with an Outlook category.



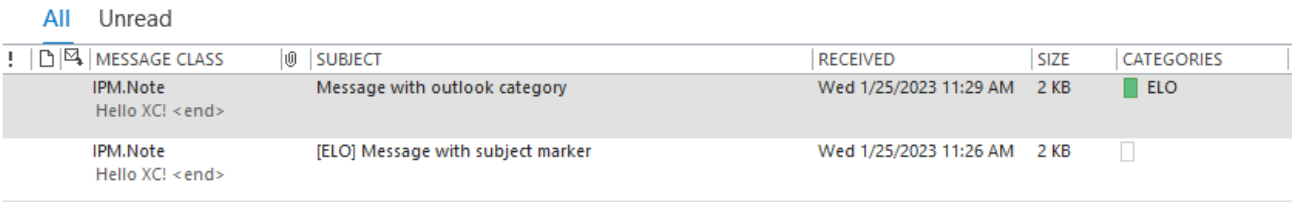
Optionally, you can specify the Outlook category when executing the *Archive* action.

If the messages are archived without tagging the subject, the following occurs:

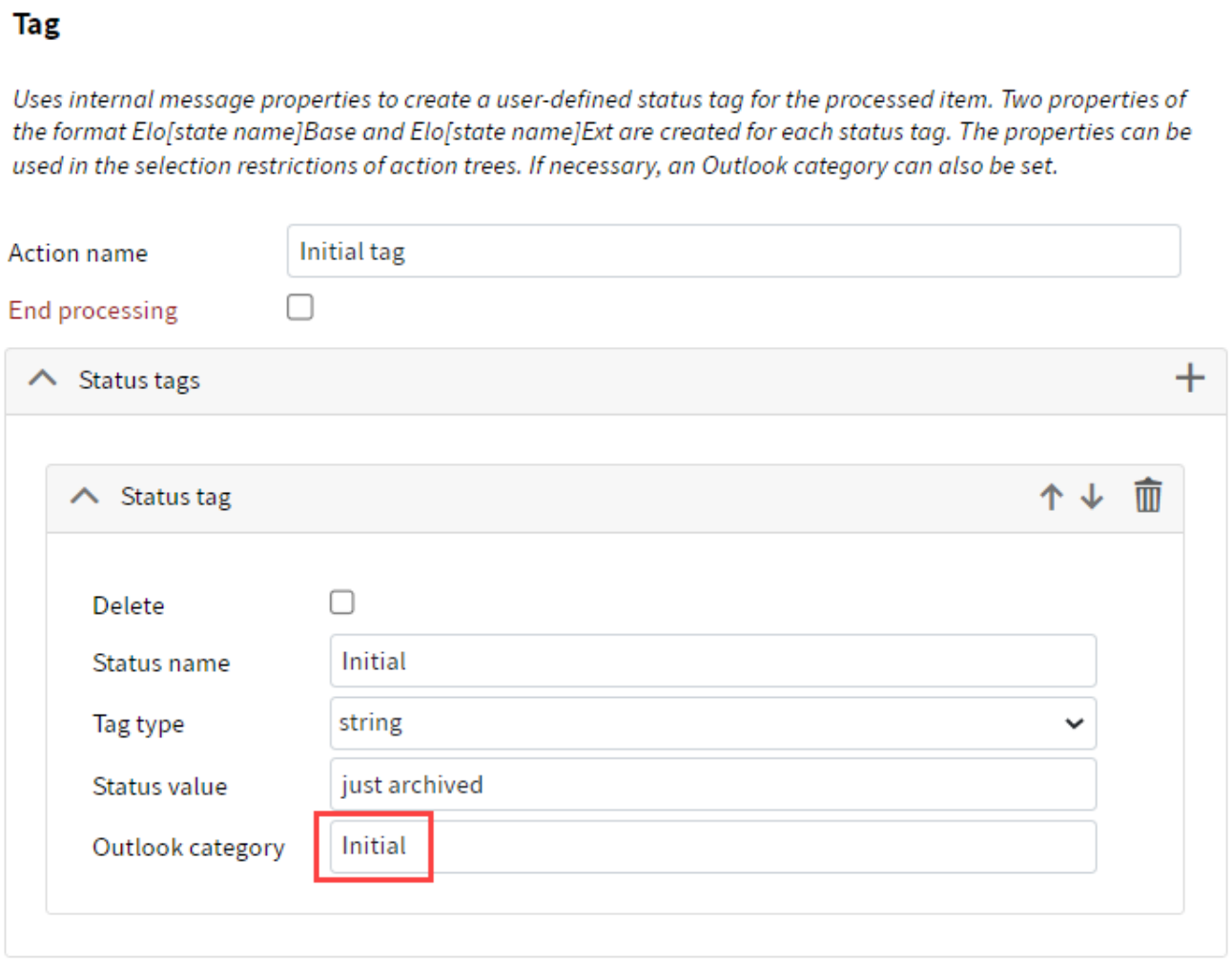


The status tag is always set internally using a property. However, the visual representation of the tag is now also different. ELO XC does not define a color category. It is either assigned randomly or

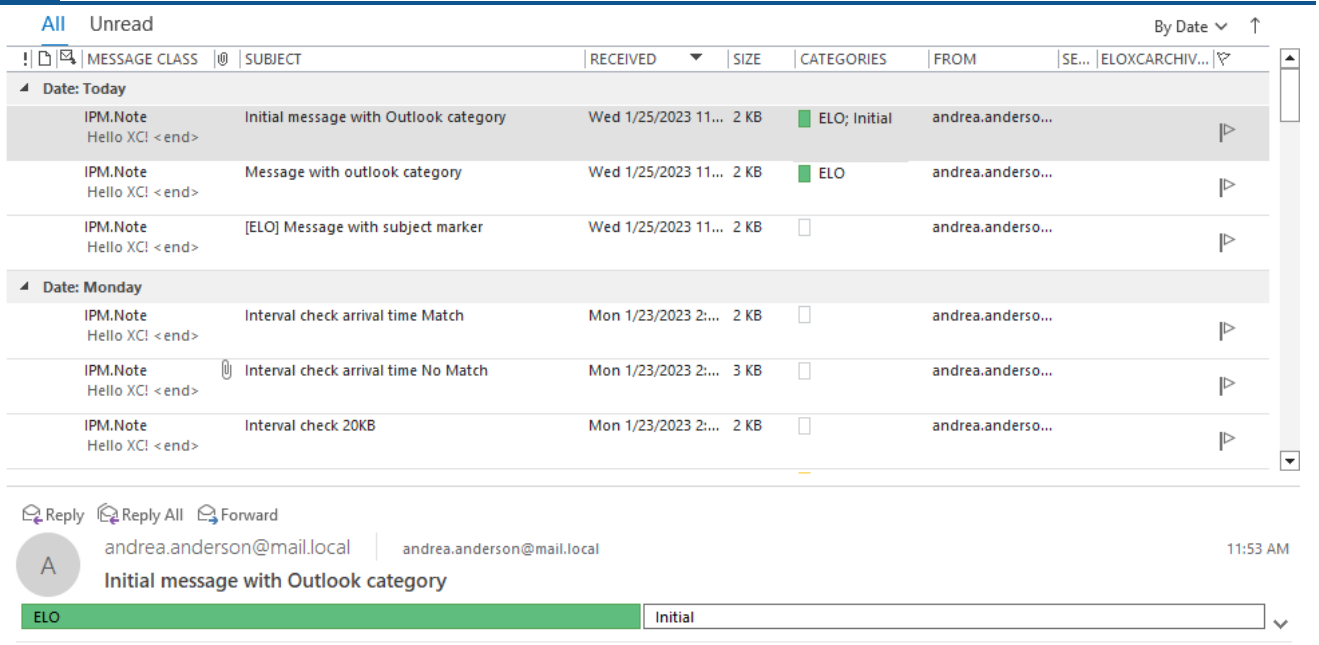
initially left empty. Once it has been selected by the mailbox owner, it is retained for the category name *ELO* going forward:



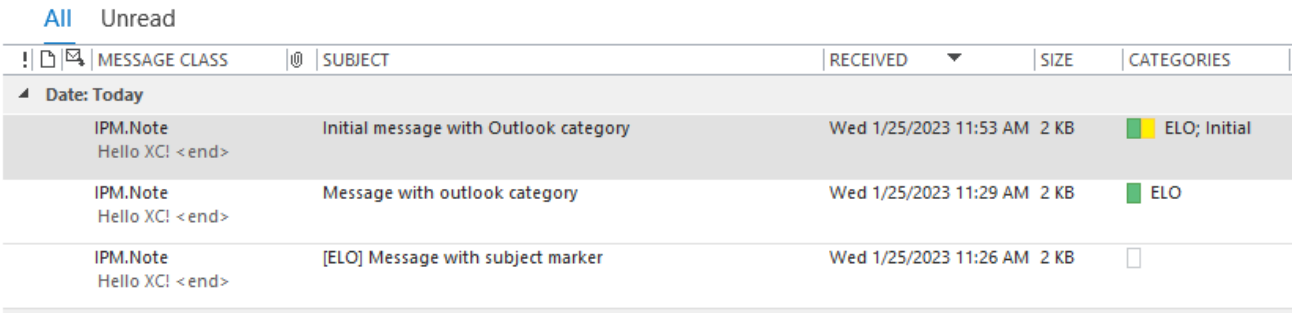
Since Outlook categories are suitable for item states, you can also use them in conjunction with status tags set by properties:



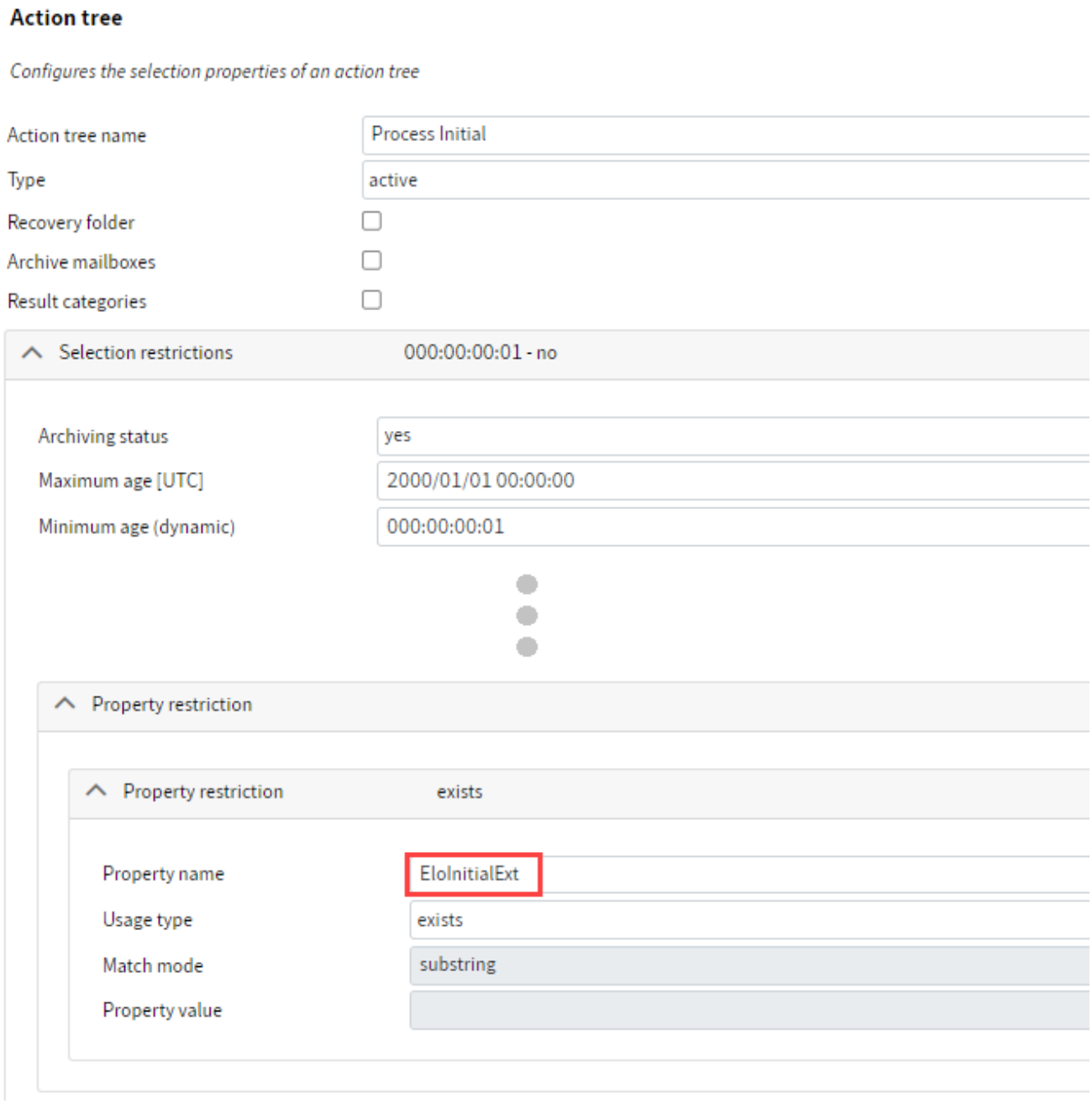
If you add this action to the action tree before saving and then file a new message, you will get the following view:



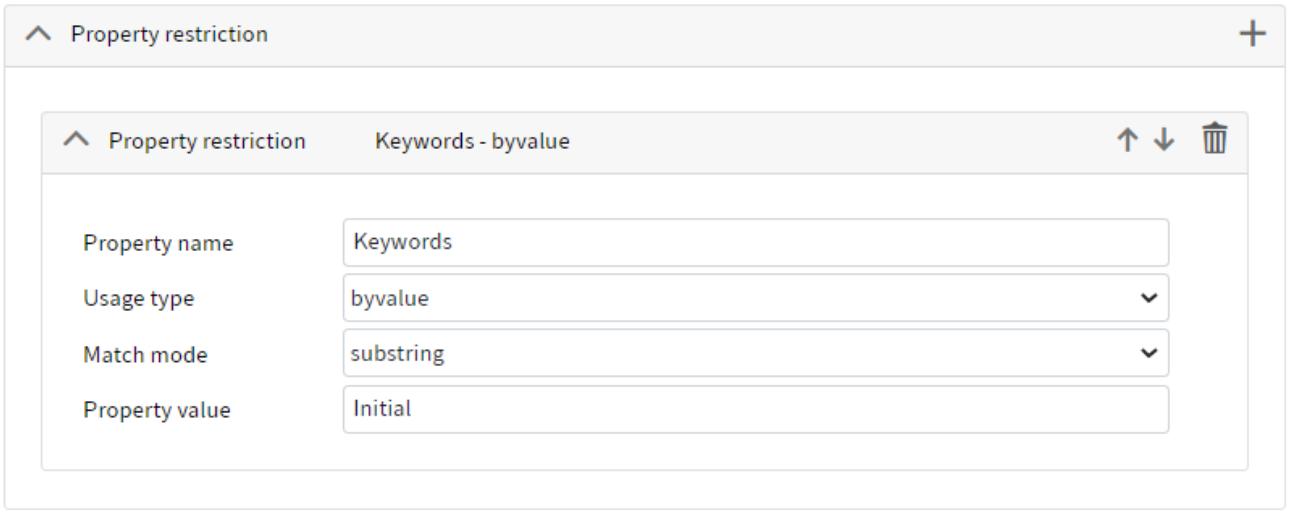
Select your own color for this category as well:



For our example, we assume that after the initial archiving, another action tree will process only those messages that were initially archived. In this case, the action tree would get this addition in the selection restrictions:

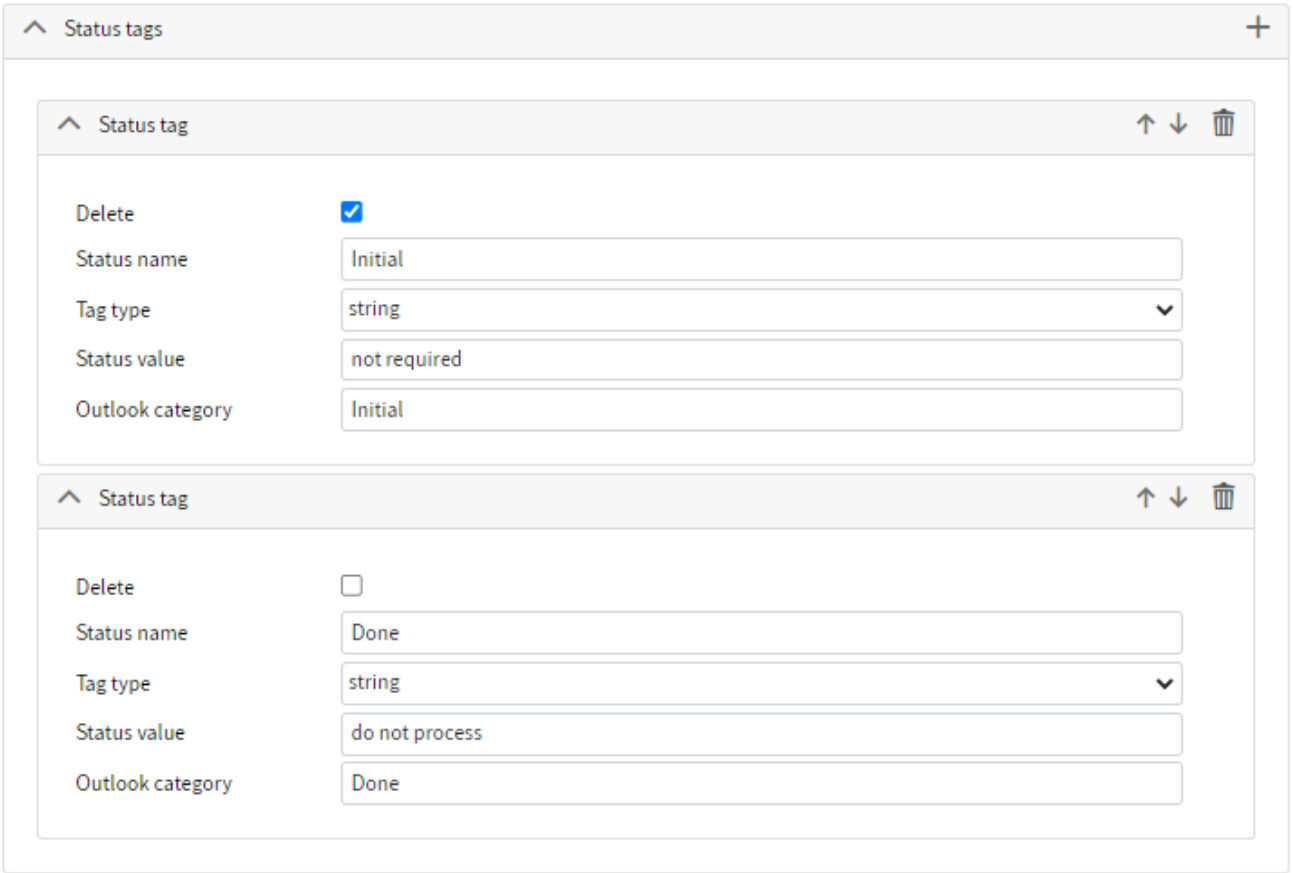


The previously set status tag is used with the automatically generated property *EloInitialExt* to select messages. This works for any archiving state, i.e. even for non-archived messages or those that are not assigned to an Outlook category. If you only want to make a selection by Outlook category, use the *Keywords* property.

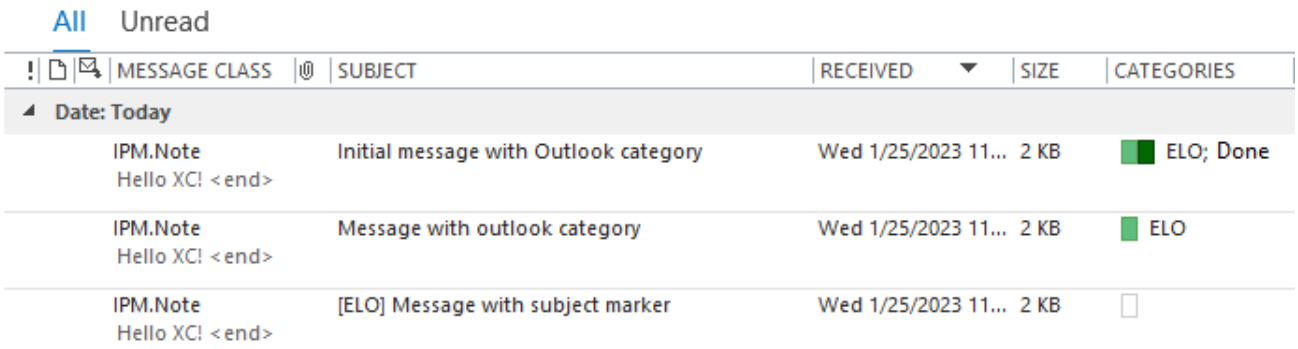


In this case, however, it is not enough to just select an Outlook category. The category name is also important and should be called *Initial*. As a result, the action tree selects only archived messages which are assigned to an Outlook category named *Initial*.

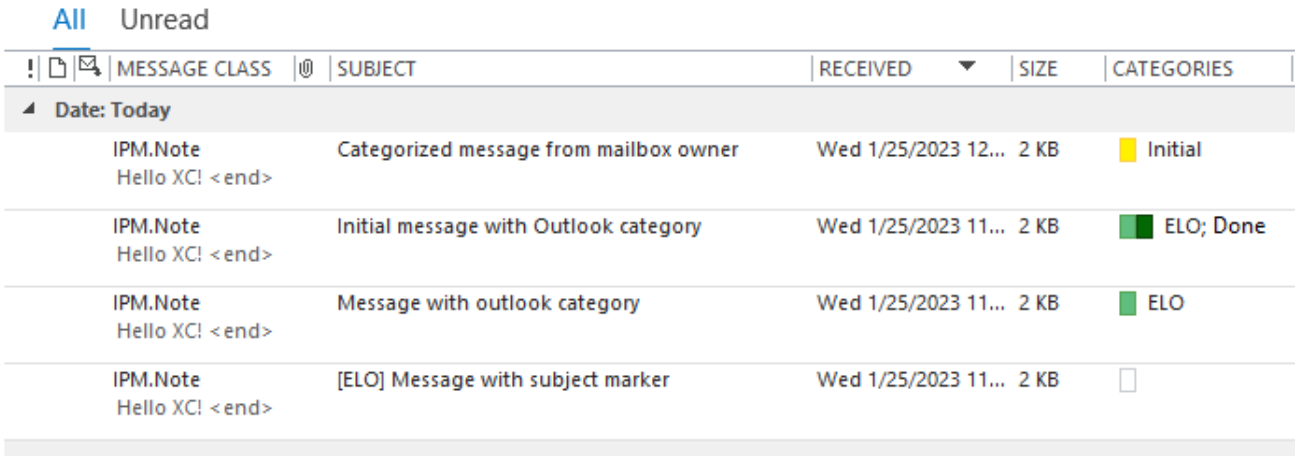
The tree in this example removes this Outlook category and sets a new category called *Done*. This requires the *Tag* action again.



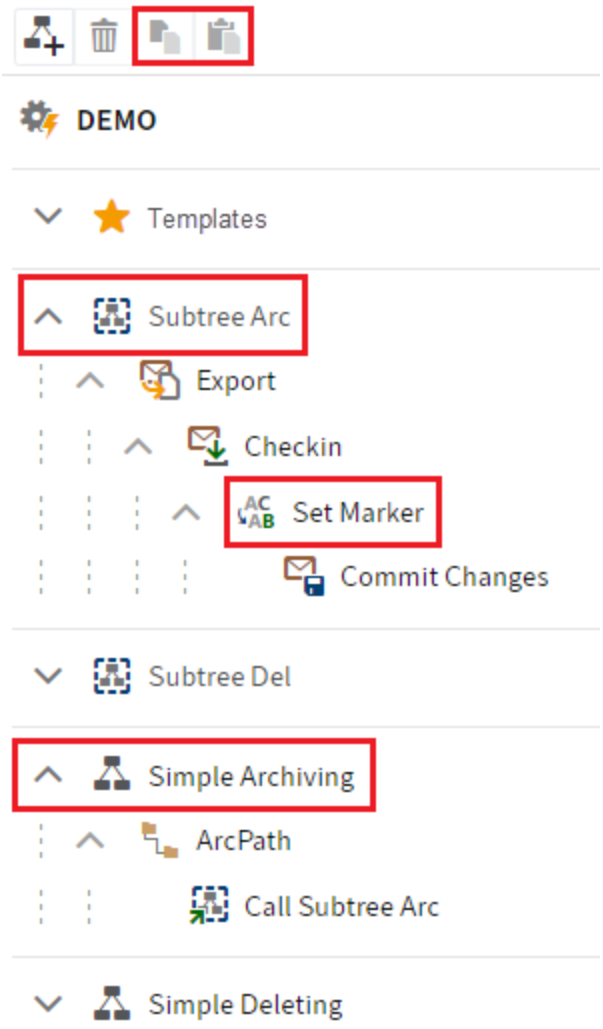
After the action tree has processed the messages, the category changes:



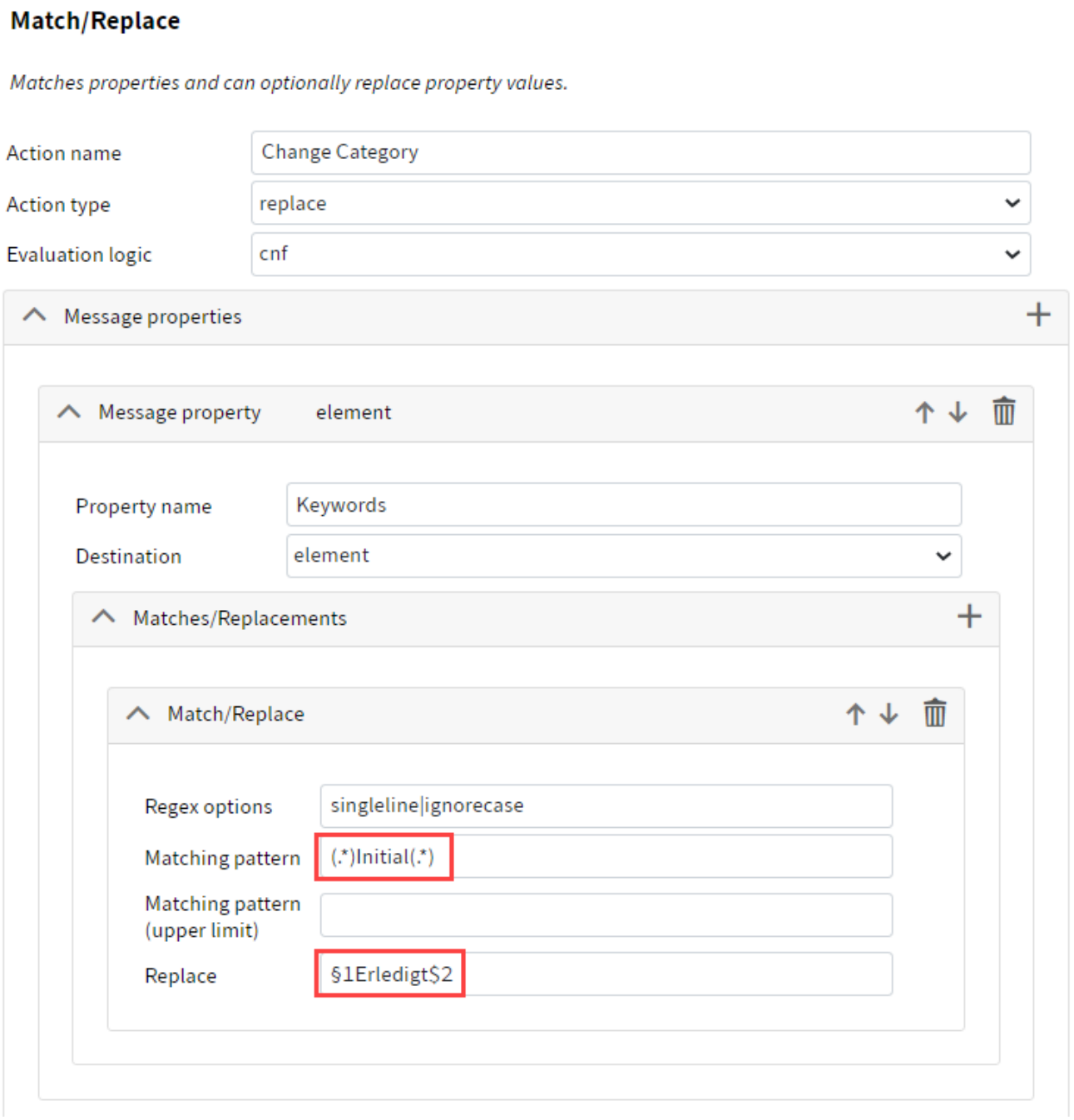
For our example, we assume that mailbox owners first categorize their messages with *Initial* before they are allowed to be processed by ELO XC. We also assume that ELO XC assigns these messages to the *Done* category after they are stored.



You now need an action tree that selects non-archived messages from the *Initial* category. To do this, you need to set the archiving status to *no* in the above example. The selection restriction to the Outlook category *Initial* remains unchanged.

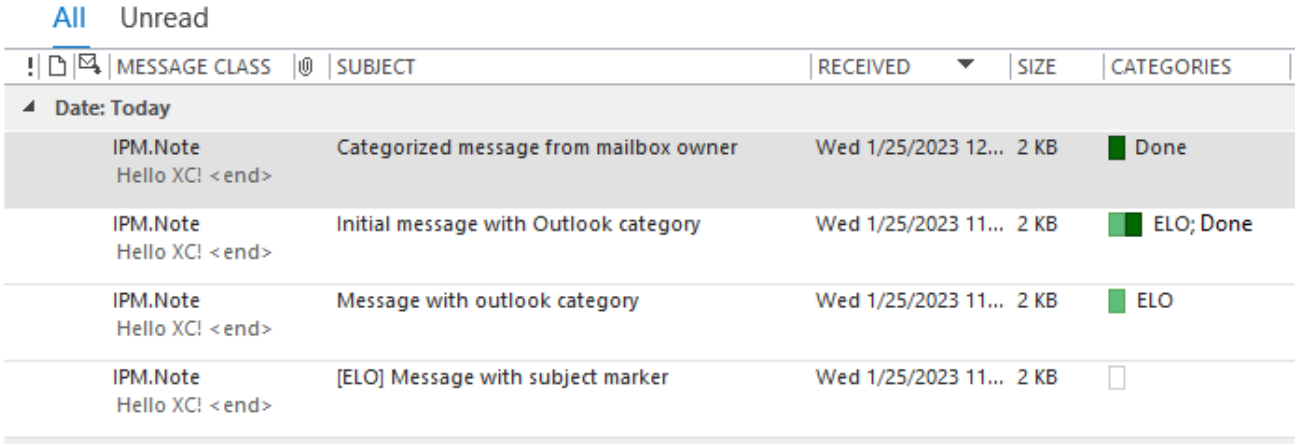


For this example, you can use the generated default configuration. Create copies of the *Subtree Arc* and *Simple Archiving* action trees. Give the copies names that allow you to identify their purpose. The active action tree is assigned the selection restriction according to the *Initial* category. In the *Set Marker* action, the subtree should change the category of the message to *Done* instead of the subject. The *Archive* action does not use an Outlook category.



The matching pattern must be chosen so that it recognizes the category name as part of a CSV list and does not change the other category names in this list. This is necessary because Outlook categories belong to the list types. These types are generally not supported by ELO XC. Therefore, the categories are kept internally as a CSV list during processing.

After the new action tree has processed the messages, the message is displayed as follows:

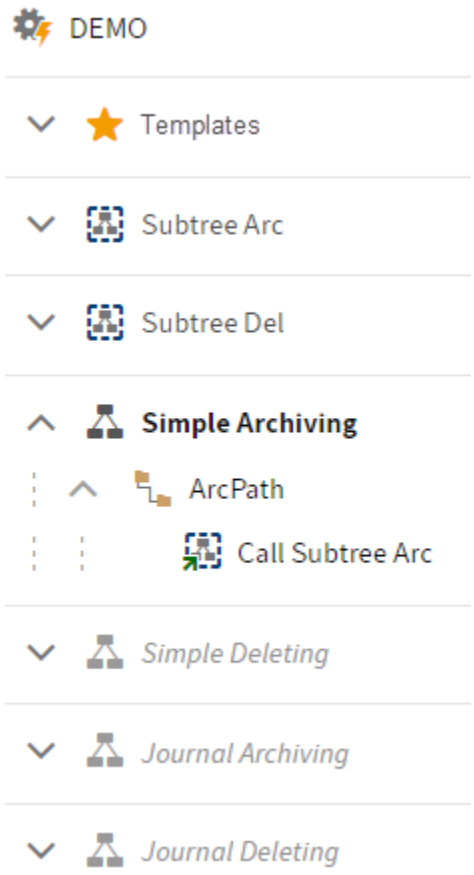

All
Unread
MESSAGE CLASS
SUBJECT
RECEIVED
SIZE
CATEGORIES
Date: Today
IPM.Note
Hello XC! <end>
Categorized message from mailbox owner
Wed 1/25/2023 12...
2 KB
Done
IPM.Note
Hello XC! <end>
Initial message with Outlook category
Wed 1/25/2023 11...
2 KB
ELO; Done
IPM.Note
Hello XC! <end>
Message with outlook category
Wed 1/25/2023 11...
2 KB
ELO
IPM.Note
Hello XC! <end>
[ELO] Message with subject marker
Wed 1/25/2023 11...
2 KB

Selection restrictions

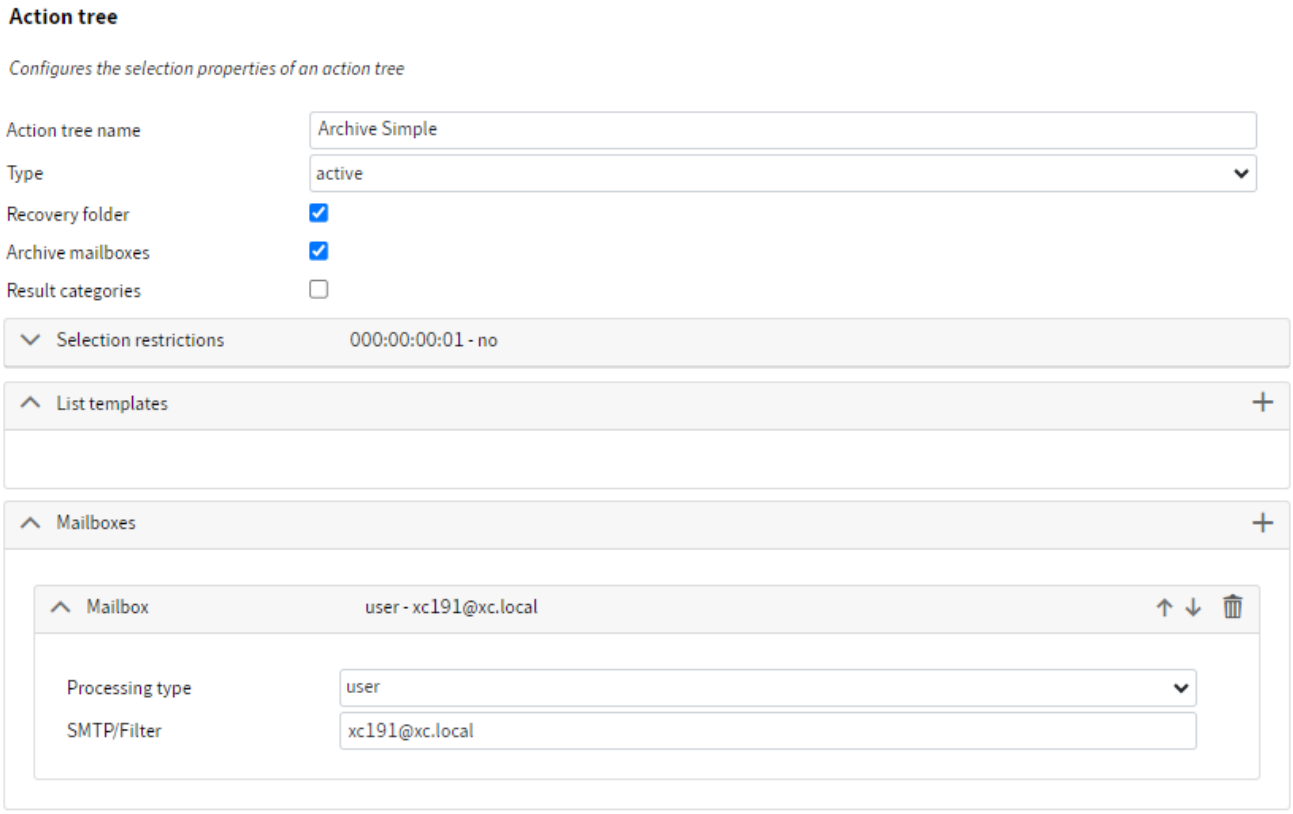
In this section, we show you examples of how to create restrictions that apply when the action trees select messages for processing. The resulting selections are the messages that are subsequently processed by the action tree.

The examples and explanations build on each other. This way, if you repeat examples at some point, you will find it easier to get started.

In addition to ELO XC, you will need Outlook and a mailbox for creating the test messages. The examples can be used for on-premises Exchange servers and Microsoft 365 Exchange Online. We recommend working with an on-premises Exchange server, as this is reduces the processing time by a factor of 25-50, which is useful considering that you have to start jobs again in a number of the examples.

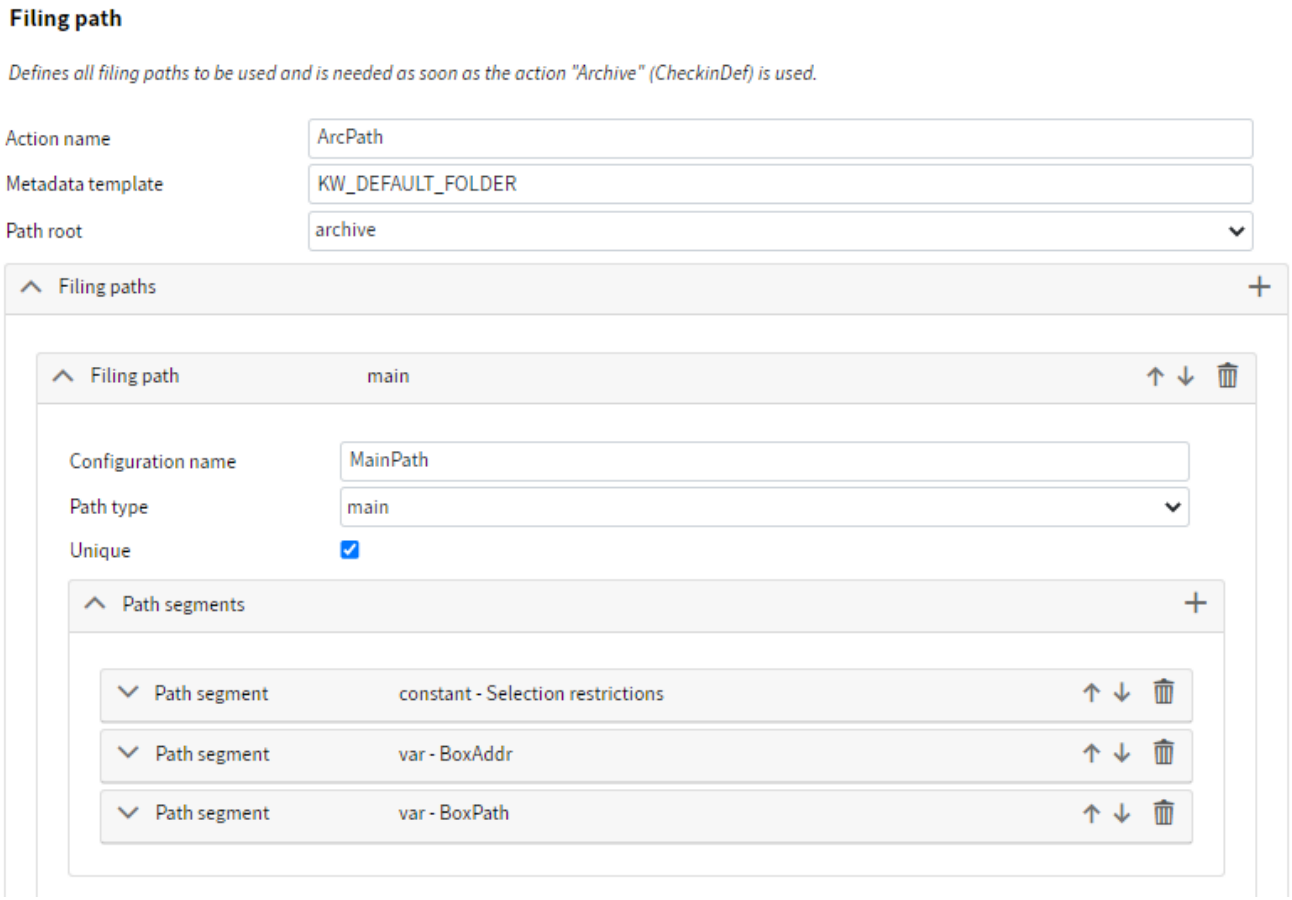


We only use one action tree, e.g. the *Simple Archiving* tree that was generated on automatic instance creation. We make changes to the tree as we go along. If you already have experience with ELO XC and have created several action trees yourself, you can reproduce these examples in one of your test instances with your own action trees. All you need to do is take note of the parameters in the action tree.



First, delete all list templates and configure only the user mailbox you are going to use.

Set the minimum age to *000:00:00:01* so that you can do the tests without having to wait. If you need other minimum ages, we show you how to configure this in the examples. Entry templates, folder filters, and message classes remain unchanged. The examples do not use folder filters, and with a few exceptions towards the end of this course, you will only create simple, small e-mails in the inbox.



Now change the *ArcPath* filing path action to make it easier to find the results in the repository. We use the main folder *Selection restrictions* in this example.

The examples of the different selection criteria are divided into subsections. You could also change the storage path in the repository for each section to get all the results afterwards. However, the examples are easier and quicker to reproduce if you don't do this.

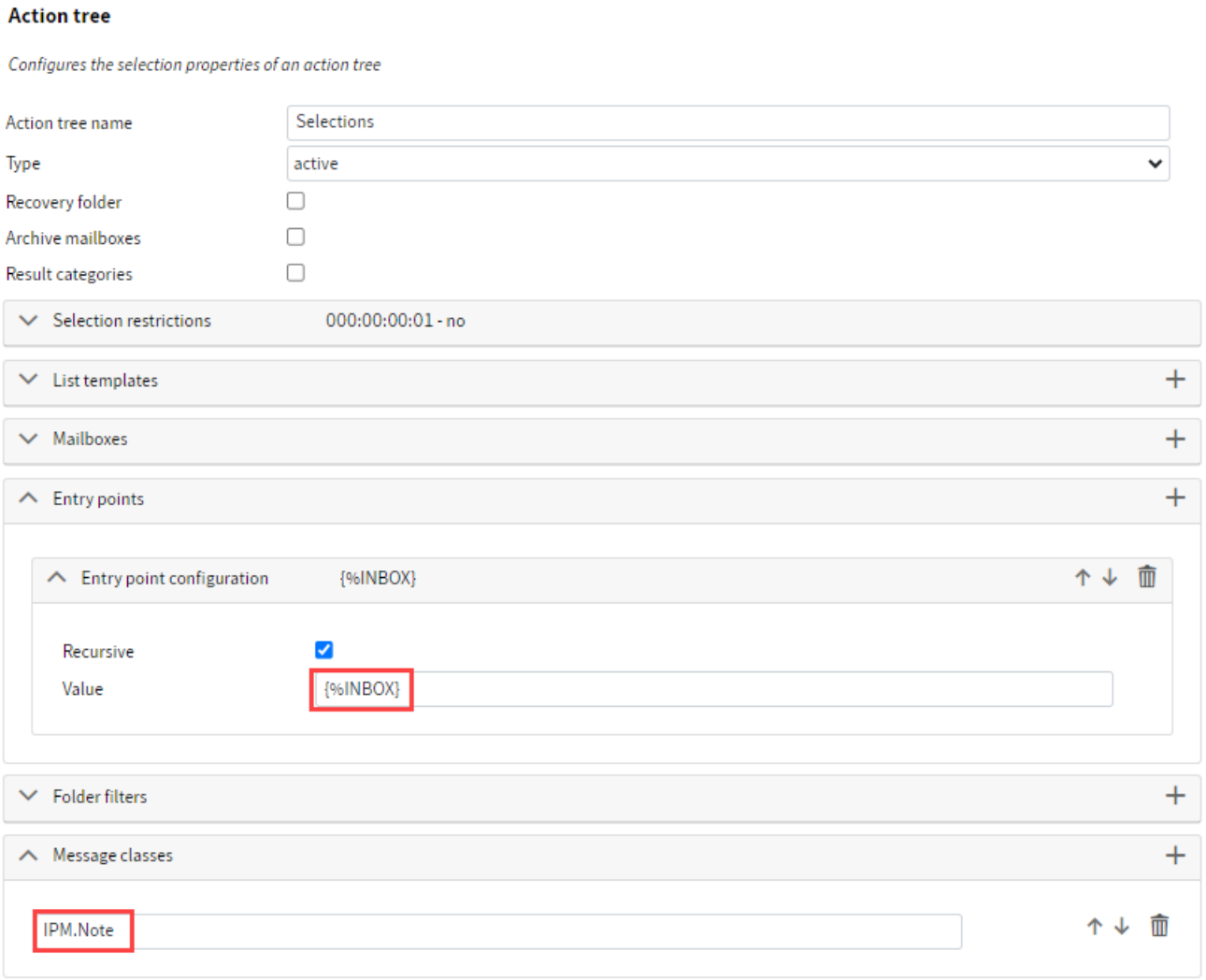
Now go back to the *Simple Archiving* action tree and change the name to *Selections*, for example. Save your changes.

Required restrictions

There are different ways to apply selection restrictions to action trees. The selection of mailboxes in itself is a type of selection restriction. Messages from mailboxes that are not configured for an action tree are not processed, which definitely constitutes a message selection restriction.

Entry points and folder filters allow you to include and exclude different mailbox folders. This is also a restriction because only messages are selected from allowed mailbox folders.

Message class selection is also a restriction because it allows you to separate folder items by the message class.

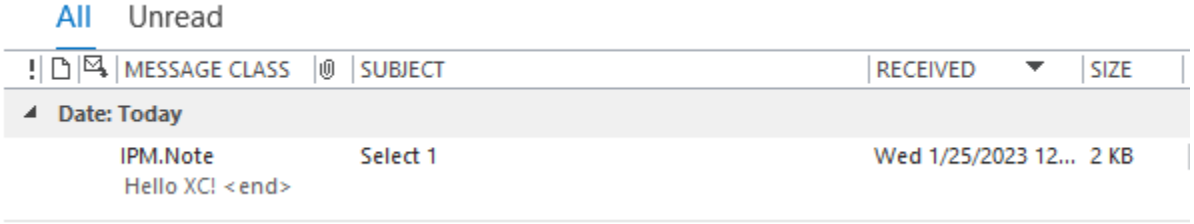


Let's assume you only want to process standard e-mails in the inbox or underlying folders. You only need an entry point and the message class *IPM.Note* (standard e-mails).

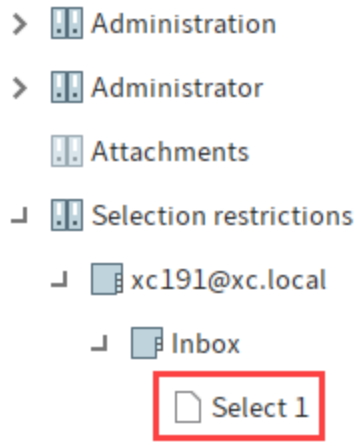
Now save the instance configuration and validate it. There should be no validation errors. Go back to the instance overview and publish your new configuration. The *unpublished* tag should have disappeared. You will use the *Change*, *Save*, *Validate*, and *Publish* steps frequently in this section. We will assume that you do this independently so that we can focus entirely on the examples.

Archiving status

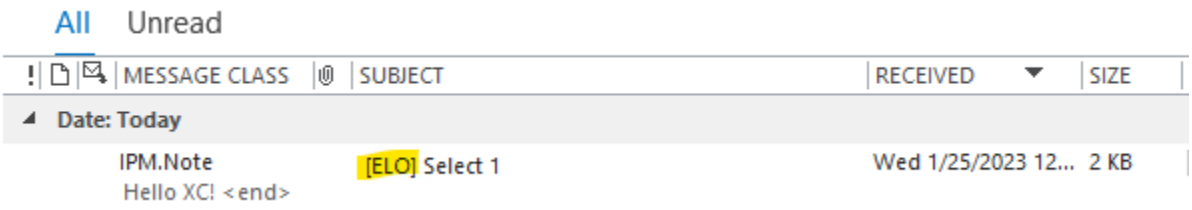
Now create any e-mail you like:



Start the instance. After a few moments, you should be able to see the new e-mail in the repository.

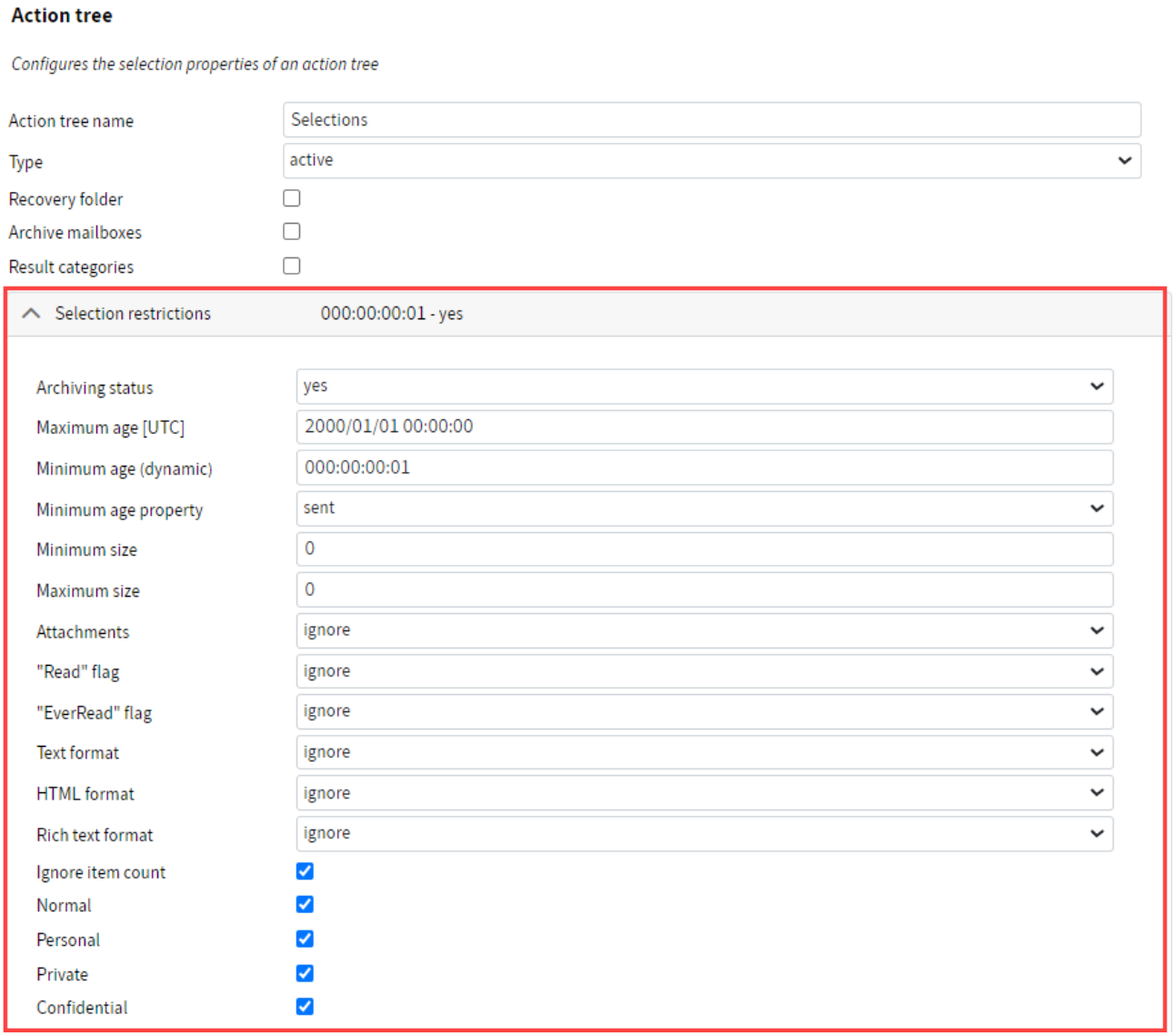


You will also notice that the subject in the ELO XC mailbox has been changed:



As you try out the examples in this section, you will repeatedly make minor changes to and publish the configuration, create a new e-mail, process it, and then see changes in the mailbox and repository. We recommend that you review the results once processing is complete.

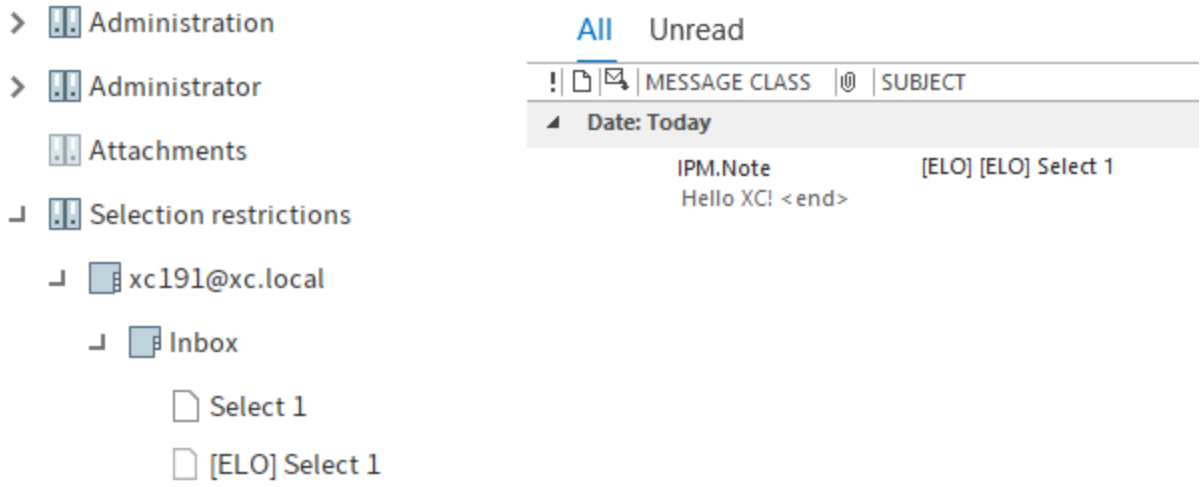
The section mainly covers selection restrictions without property restrictions. The latter allows you to apply custom properties, another important part of message selection, but in this case we only go into detail on the standard selection criteria without custom extensions:



You have already used the archiving status. although it occurred automatically. It is set to *no*, which means that ELO XC will only process messages that have not yet been archived, i.e. stored in the repository. Now start the instance again and check if the *Select 1* message is stored. It should show up as a duplicate in the repository, but that doesn't happen because the archiving status prevents that from happening. You can now repeat the process as often as you like without *Select 1* being processed again.

This is because the message was assigned an internal tag, known as the archiving status, the first time it was processed. This is how ELO XC marks elements that have already been processed, i.e. stored in the repository. This corresponds to the vast majority of all use cases, since message duplicates are not usually required. However, you can store the message a second time if needed.

Change the archiving status to *yes*. After saving, validating, and publishing, run the instance again.



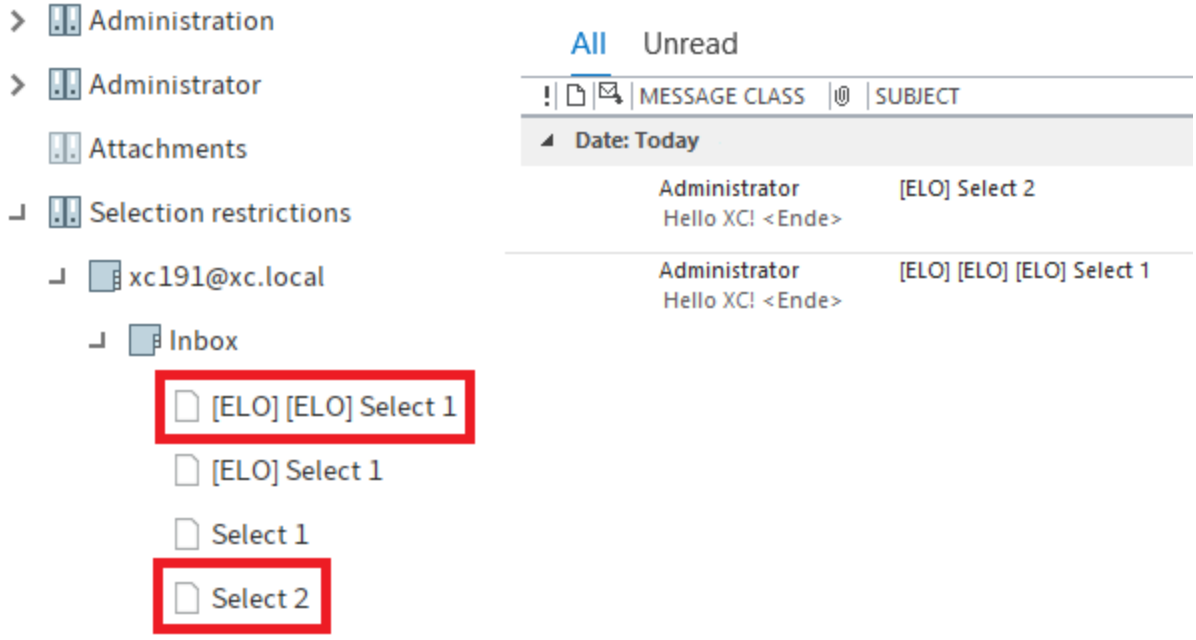
The message shows up in the repository a second time and is marked as a duplicate in the mailbox.

Of course, changing the archiving status to *yes* in the selection restrictions ensures that only those messages that already have the archiving identifier are now selected for processing. The fact that the selected message shows up in the repository is due to the fact that ELO XC always processes currently valid items. If a message has a subject tag during processing, it is filed with that subject. Tagging it again means that it is duplicated in the mailbox. Later you will learn to tag messages differently, but the default behavior is sufficient for this case.

Change the archiving status to *ignore*. What effect will this have? And how would you go about checking this?

You need a second message that has not been archived yet. Create an e-mail with the subject *Select 2* and start processing. Did you expect what happened? You can reproduce this example on your own before reading on.

You should get the following result:



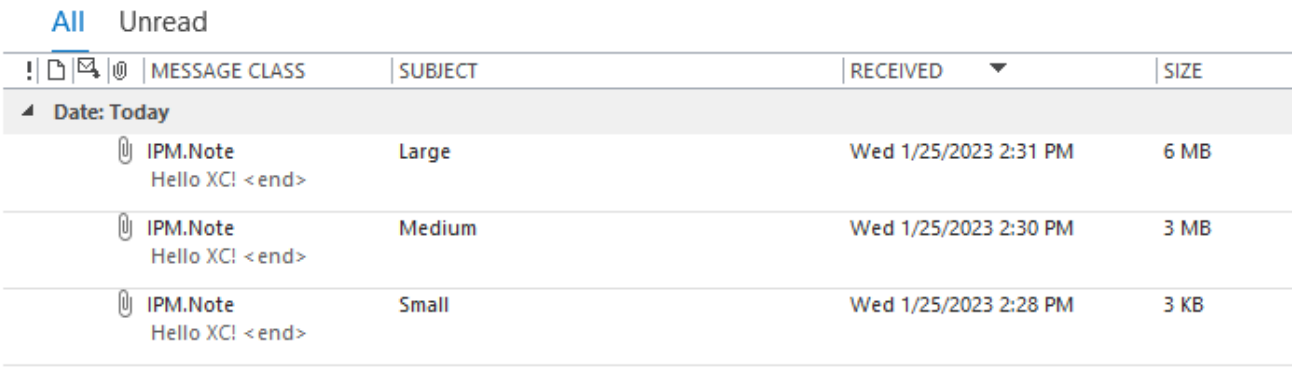
The messages are processed regardless of their archiving status, both the new message *Select 2*, which currently has no archiving identifier, as well as *Select 1*, which has already been processed twice.

Delete the stored copies of *Select 1* and *Select 2* in the repository and mailbox.

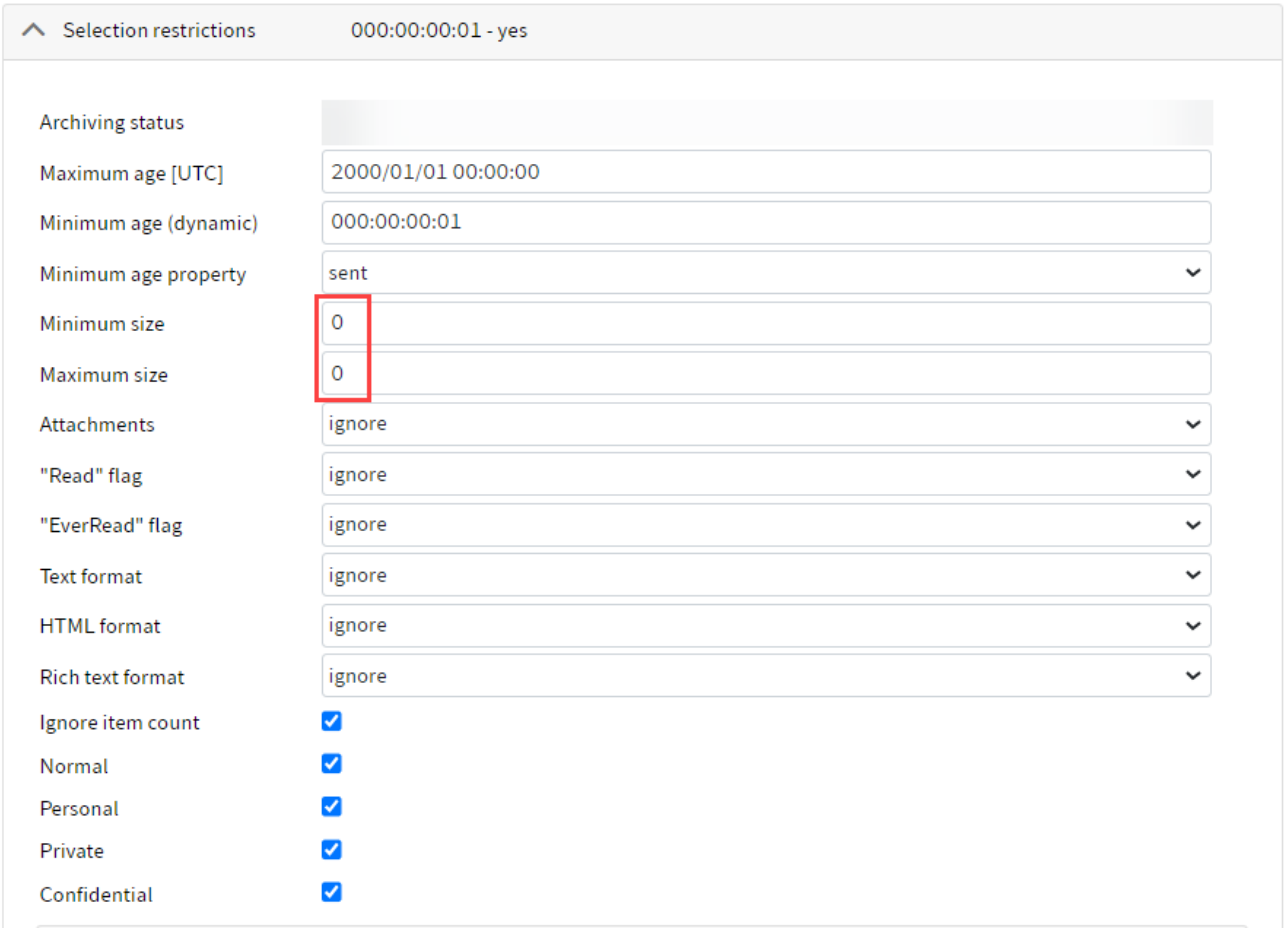
> **Information**
>
> For all other selection examples, we use only the *ignore* status in this section. In practice, this is rarely necessary, but usually only when checks, categorizations, or deletions are needed regardless of archiving status and without filing the items. For our examples, on the other hand, it is useful to select this status as it means we do not have to recreate test messages with each processing run. We accept that there are duplicates in this case.

Size restrictions

Pick three attachments (e.g., small images or PDF files) with significantly varying file sizes. Attach the smallest one to the *Small* message, the medium-sized attachment to the *Medium* message, and the largest one to the *Large* message. You should see the following messages in Outlook:



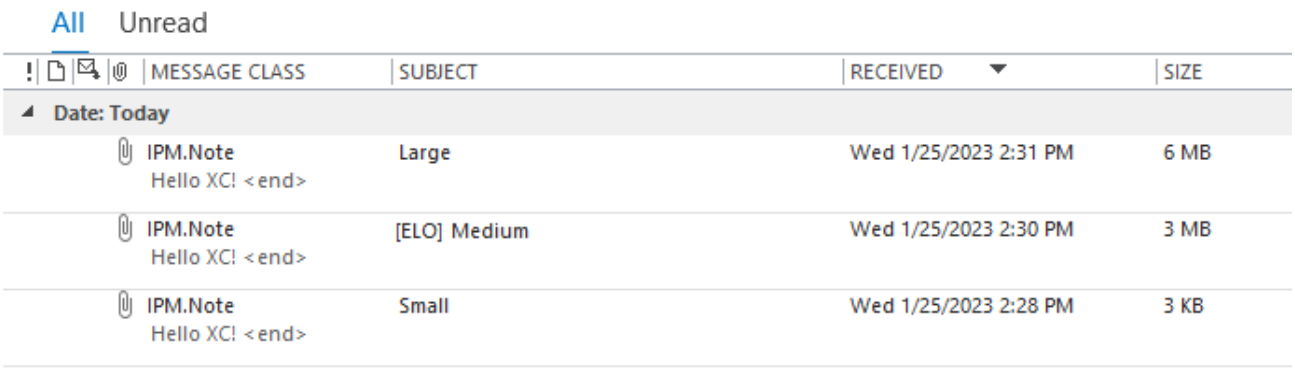
Let us assume that you want to process the medium-sized message with a size restriction. How would you go about it? Take a quick look at the selection criteria (1 kilobyte is approximately 1000 bytes).



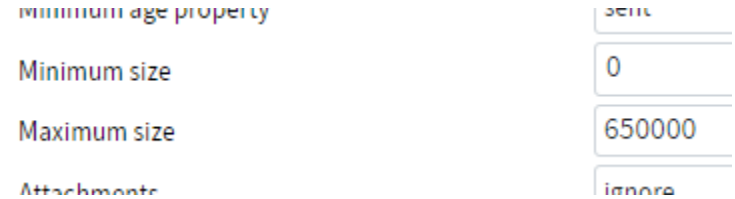
The messages can be filtered by their total size (all message parts, including file attachments).

To process the medium-sized message in the example shown, a minimum size of 300000 bytes and a maximum size of 400000 bytes is sufficient. You need to determine the size of your medium-sized message in Outlook. The following applies for our example:

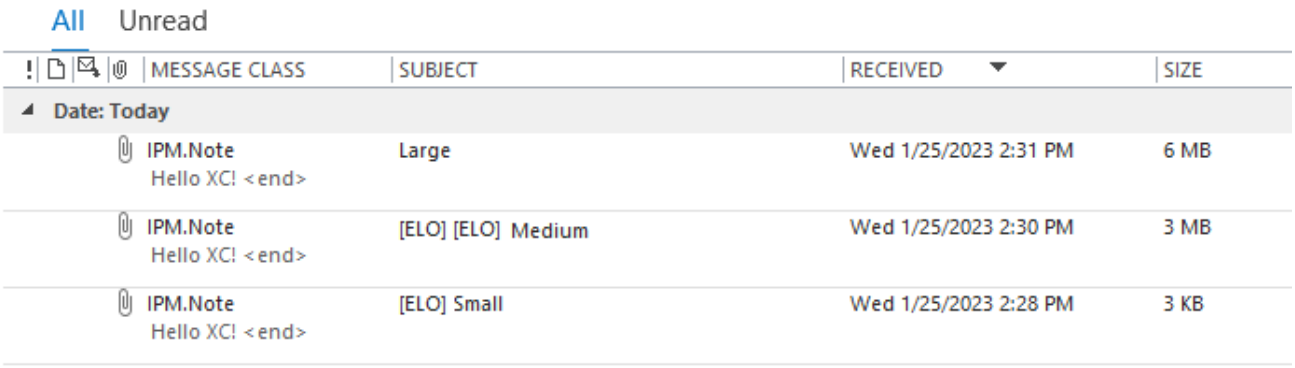
After processing, your *Medium* message should have a tag and the original should be in the repository. Is this the case?



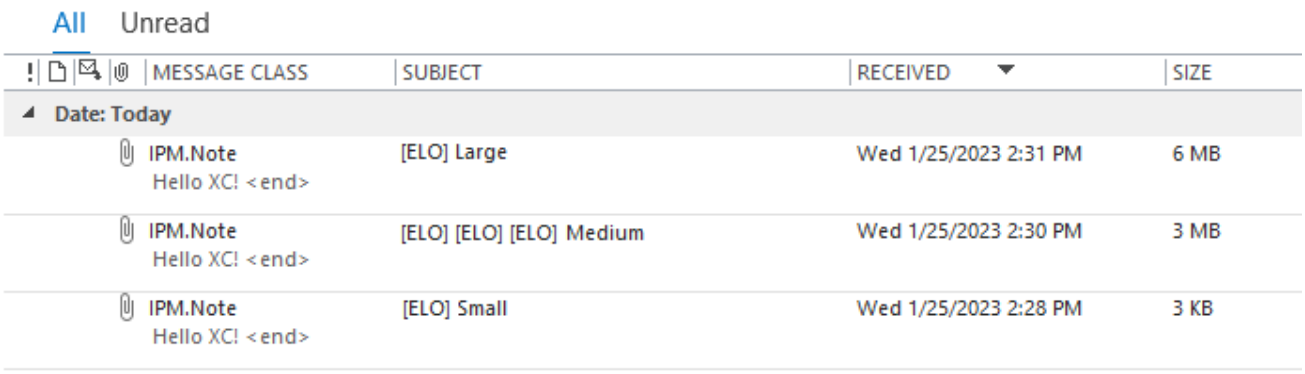
Now find a size limit so that all messages except *Large* are processed. Keep in mind that in a real-life scenario, this could affect a lot of e-mails. How would you select the limit if you assume that you currently see only the three most recent messages in your mailbox, but in fact there are many other messages that this size limit should apply to?



Since there can be any number of messages smaller than *Large*, and also very small ones, the lower limit is set to `0`. In the mailbox, you can see that *Medium* has been processed again, which is what we had intended:



Repeat the test with a configuration that now processes all messages except *Small*. Of course, you now already know what result to expect. As you can see, this is also the result you get.



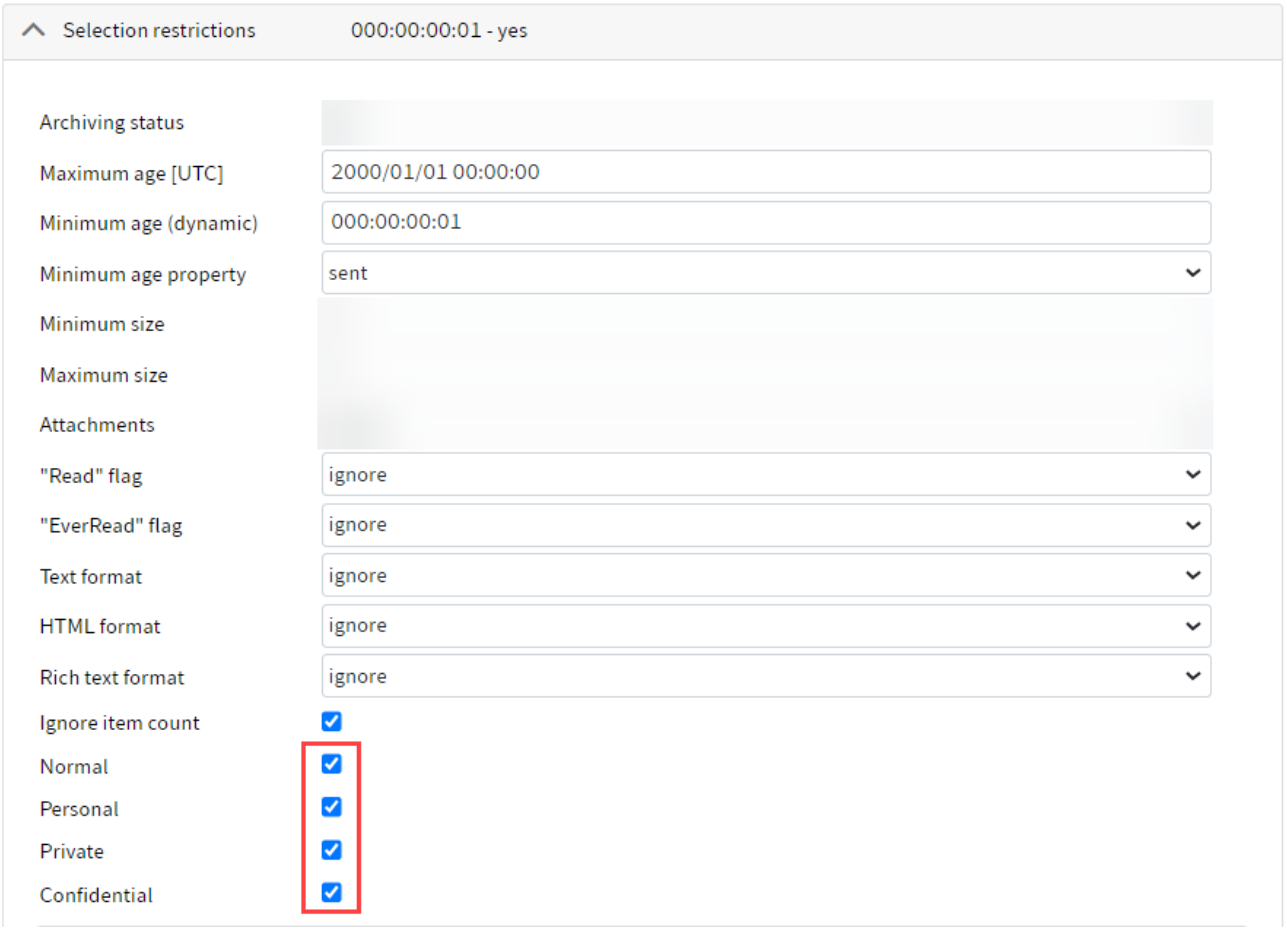
> **Information**
>
> You have to assume that even minor changes to the configuration do not always have a noticeable effect. ELO XC offers you a total of about 500 configuration parameters for individually processing and storing messages. Many of these parameters are rarely used and are useful only in certain cases. Therefore, you cannot expect to know all parameters and to predict their effect in every case, but you can always check how they work. As you progress in ELO XC, you will learn which changes you need to check and which you can assume will function.

In the last configuration, did you consider that there might be messages even bigger than *Large*? The configuration should be approximately like this:

Above *Small* is the minimum size of the selection. With a maximum size of `0`, the upper limit is ignored or not used. In this case, all messages that are larger than the minimum size are selected. Clean up your mailbox again as well as the repository. Remember to set the size limit back to `0`.

Attachments

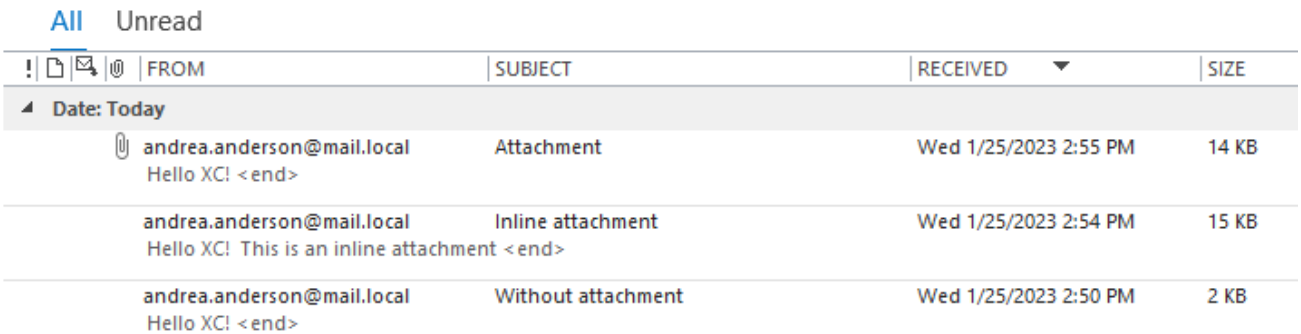
The parameter values *ignore*, *no*, and *yes* are used several times in the selection restrictions, e.g. also for the selection restriction by attachment.



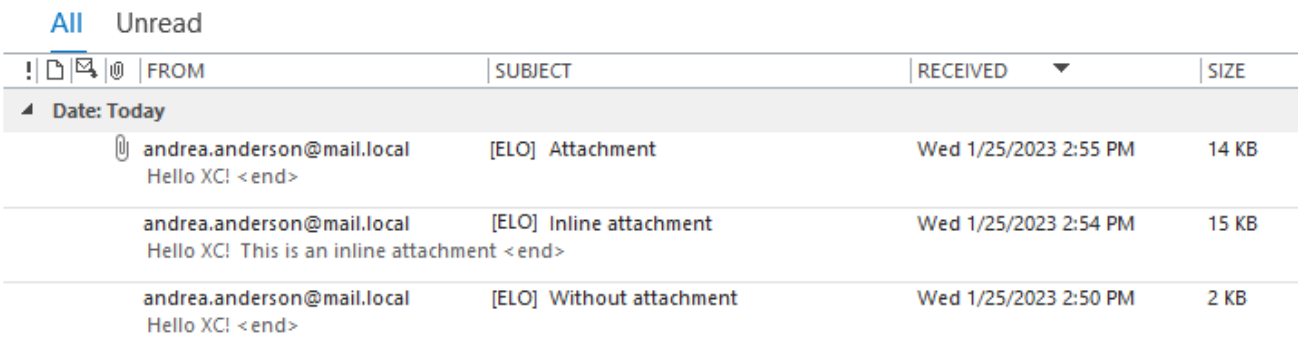
Now create these three e-mails:

- *Without attachment* should not have any attachments.
- *Inline attachment* should not contain an attachment, but an embedded image. You can copy and paste any image you like into the body of the message. Add a small text below the image.
- While you are creating *Attachment*, drag and drop a file into the attachments.

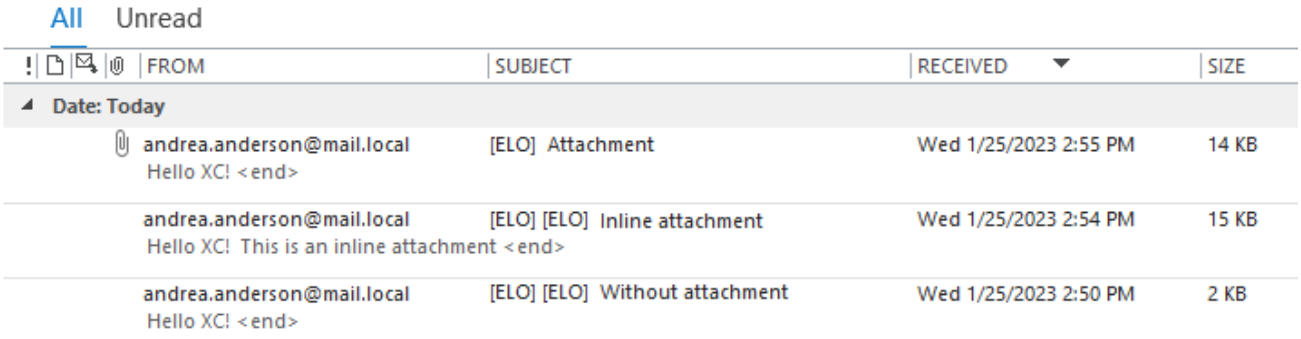
You now have these three e-mails in the inbox:



Process these messages with attachments set to *ignore*. The result is:



If you now change the restrictions so that ELO XC cannot select messages with attachments, what result do you think you would get? Try it for yourself.

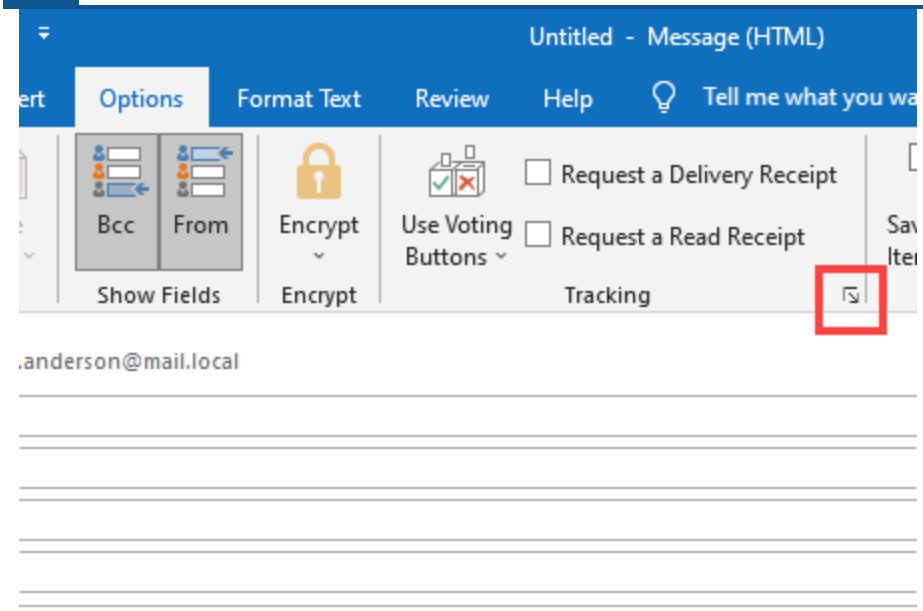


The messages with the embedded image/inline attachment are selected because they do not have an attachment. This makes sense because when attachments are extracted with the *Export* action and stored separately from the message, the message body is retained. Inline attachments are interpreted as part of the written message.

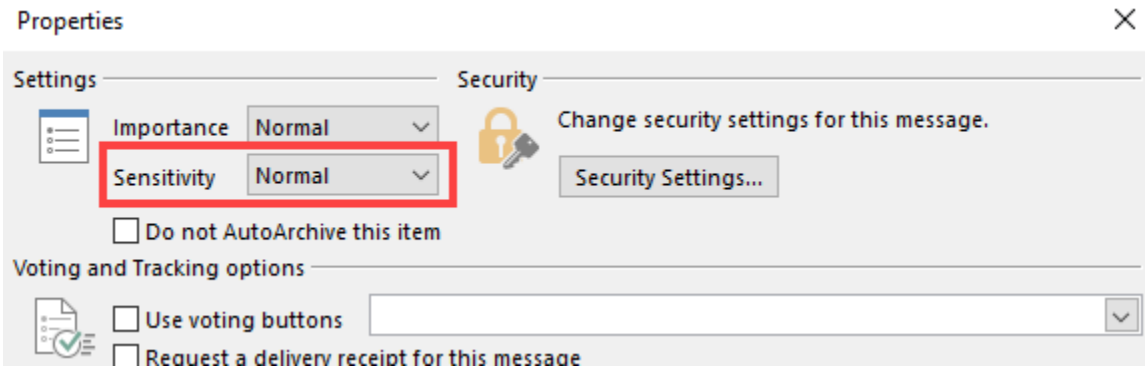
Change the restriction for attachments to *yes* and start processing again. You will now see that the *Attachment* e-mail has been processed a second time.

Sensitivity level

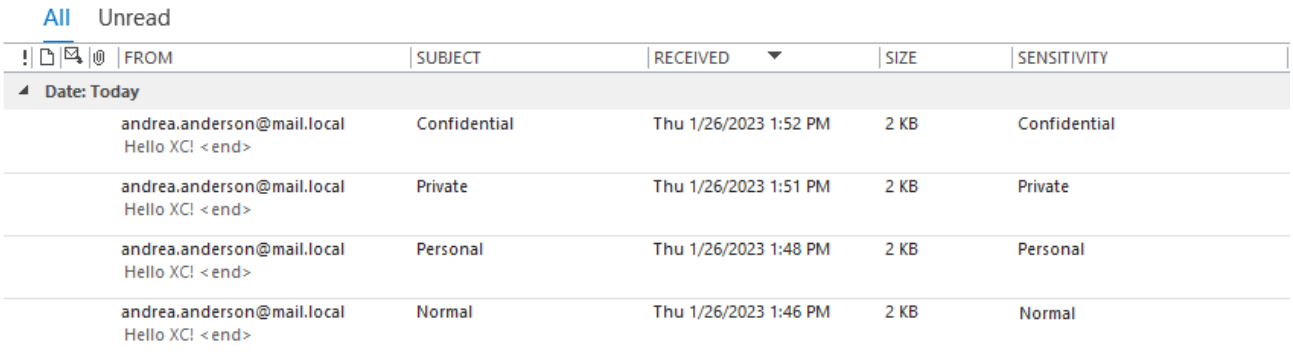
Now create a new *Normal* message. Select the *Options* in Outlook.



Open the *Properties* and select the *Sensitivity* settings.



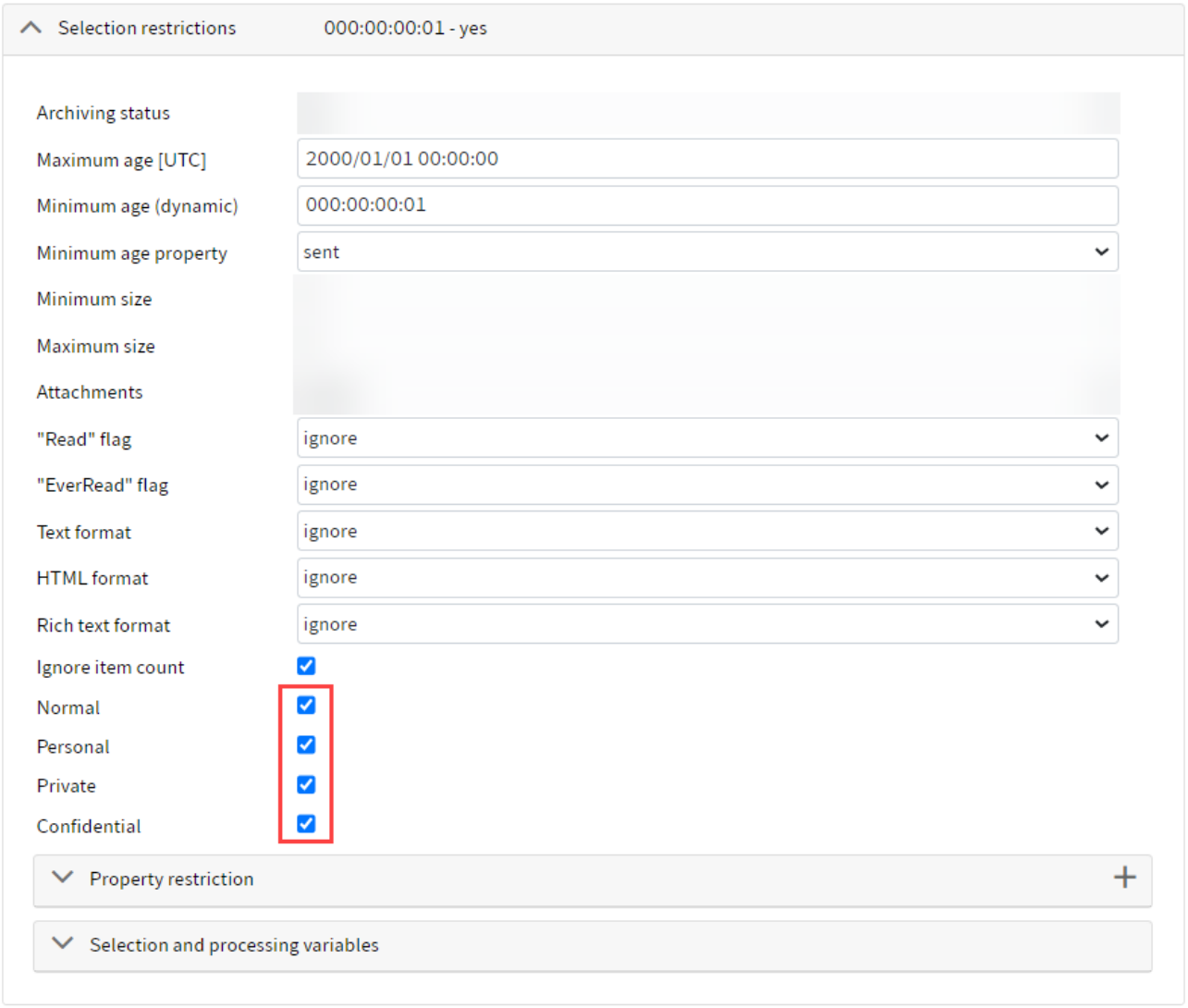
Create a message for each of these sensitivity levels You should now have four new messages in your mailbox:



Information

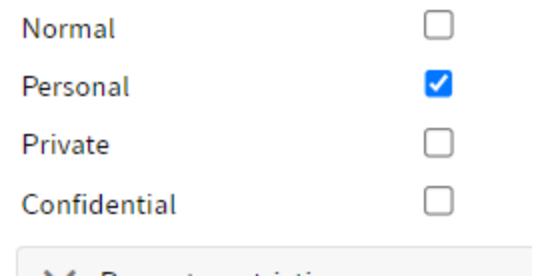
If you have created four messages according to the instructions and still only see three because *Private* is missing, it is not because you have made a mistake. The reason is probably that you are using an Outlook profile with multiple mailboxes and only one account was used to log on. The account that is used to log on is allowed to access other mailboxes by proxy. If you sent the messages from the mailbox of the logged in account (Outlook standard for new items), you will find all four messages in *Sent items* of the corresponding mailbox, while only three are visible in the inbox of the recipient mailbox. This mailbox is associated with another user account and therefore does not reveal its private messages to other accounts.

The sensitivity level selection restrictions:

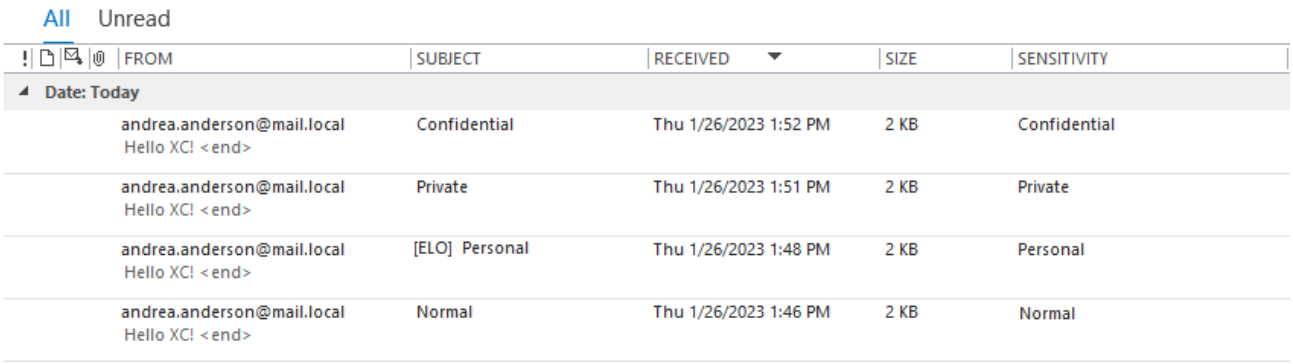


The available options are *Normal, Personal, Private*, or *Confidential*. ELO XC allows all sensitivity levels by default. If a level is disabled, messages with this sensitivity level are not selected.

We now want to process only personal messages:



We can expect this result in the mailbox:

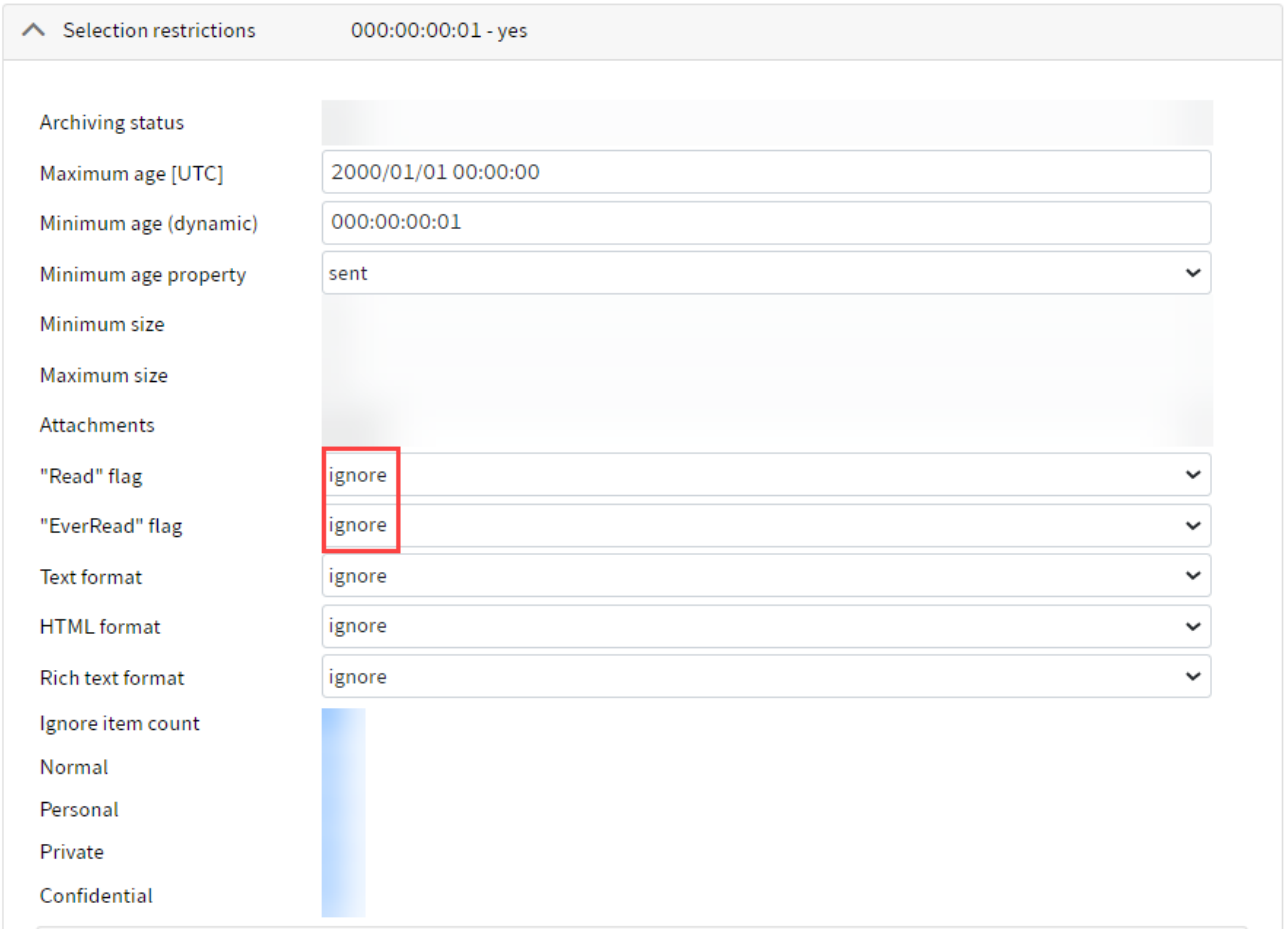


Take your time and test the other sensitivity levels, or different combinations of these levels. When you have finished, you need to clean up Outlook and the repository again and remove the confidentiality level restriction in the action tree.

Read or unread?

You are probably already familiar with the option to mark messages as *Unread* after reading. This function is a simple way to create a reminder. But are you also aware of the trick that tells you a message has already been read at least once even though it has been marked as *Unread*?

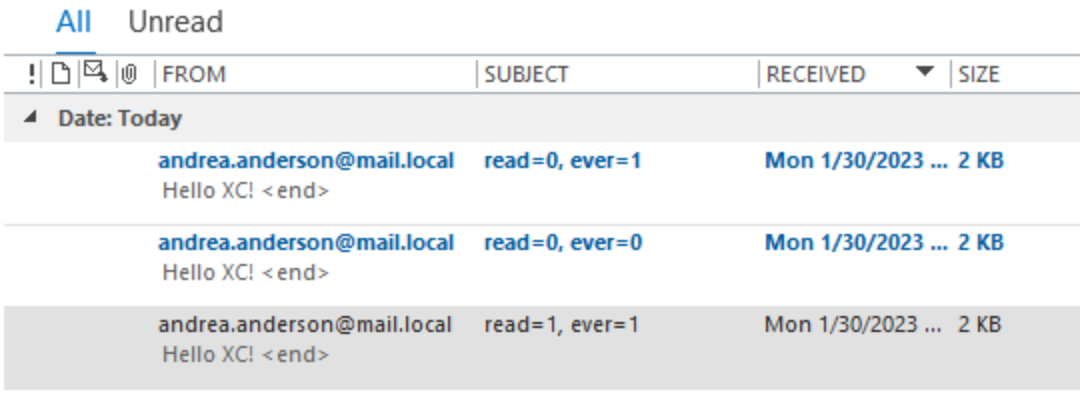
You can configure this distinction with the *Read* flag in the selection restriction.



When you are testing this, you need to be careful about how many clicks you make as otherwise you will have to do the test again. This is due to the behavior in Outlook. As soon as a message has been selected once, the *Read* flag is set and saved when you select another message in Outlook. However, the *EverRead* flag is also set. If you reset the status to *Unread*, only the *Read* flag is removed. *EverRead* is never reset.

This is how Exchange keeps track of whether a mailbox owner has noticed a message at least once. The user-side function to reset *Read* has a counter function on the server side, *EverRead*.

For this example, you need the following messages in the mailbox. Before you start, read the instructions below.



`0` and `1` indicate whether the respective flag is set. Create `read=1, ever=1`.

After the message is delivered, select it and leave it selected.

Now create `read=0, ever=0` and `read=0, ever=1`.

Next, select `read=0, ever=1` and then `read=1, ever=1`.

Since `read=0, ever=1` is now marked as *Read*, set the status back to *Unread*.

Finally, select `read=1, ever=1` again.

From this point, do not select any of the messages until your tests are complete, as it could corrupt your test data.

> **Information**
>
> There is no need to consider the combination of *Read* and *EverRead*, as it cannot occur. Therefore, the configuration *Read=yes* and *EverRead=no* throws a validation error.
>
> The following table shows which messages should be selected and stored in your tests. Test the combinations for *Read* and *EverRead*.

Read	EverRead	read=0, ever=1	read=0, ever=0	read=1, ever=1
ignore	ignore	X	X	X
no	ignore	X	X	
yes	ignore			X
ignore	no		X	
no	no		X	
ignore	yes	X		X
no	yes	X		
yes	yes			X

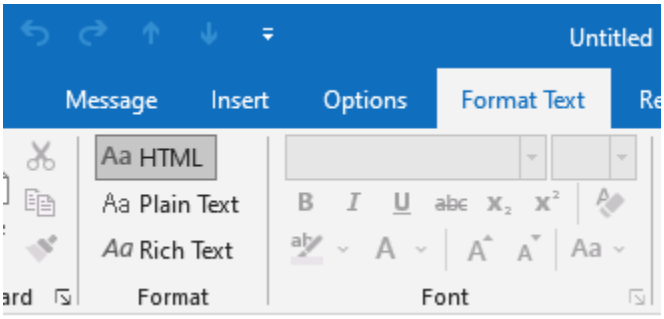
> **Information**

> The read status is an important property of e-mails. Since ELO XC works independently of mailbox owners, it is possible to make settings that ensure that mailbox owners never see messages because ELO XC has already stored them in the repository and removed them from the mailbox before the mailbox owners access their mailbox again. You should consider this if mailbox owners are unable to read their e-mails for a longer period (e.g. because of vacation). In a journal mailbox, however, message selection based on the read status would be an obstacle.

Body format

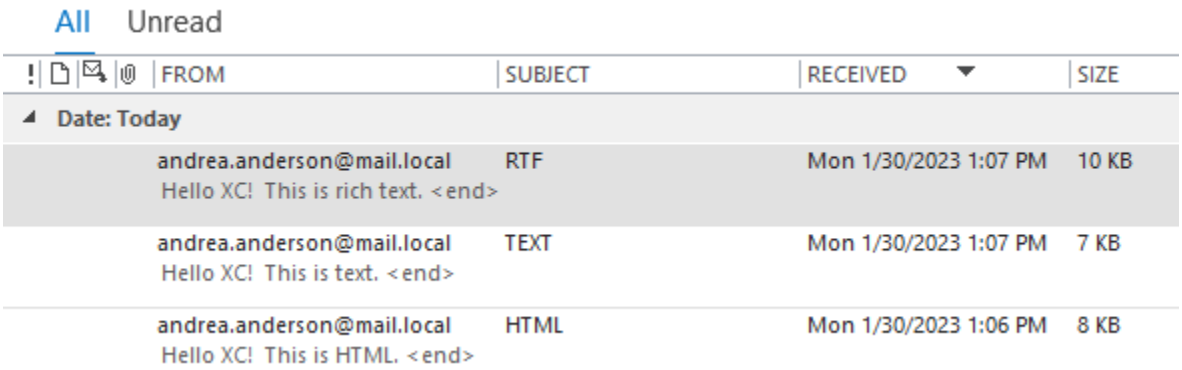
Most e-mail clients support text format and HTML format. Outlook additionally supports RTF, although this format is outdated.

Create one message each with the subject *HTML*, *TEXT*, and *RTF* and use the appropriate format:

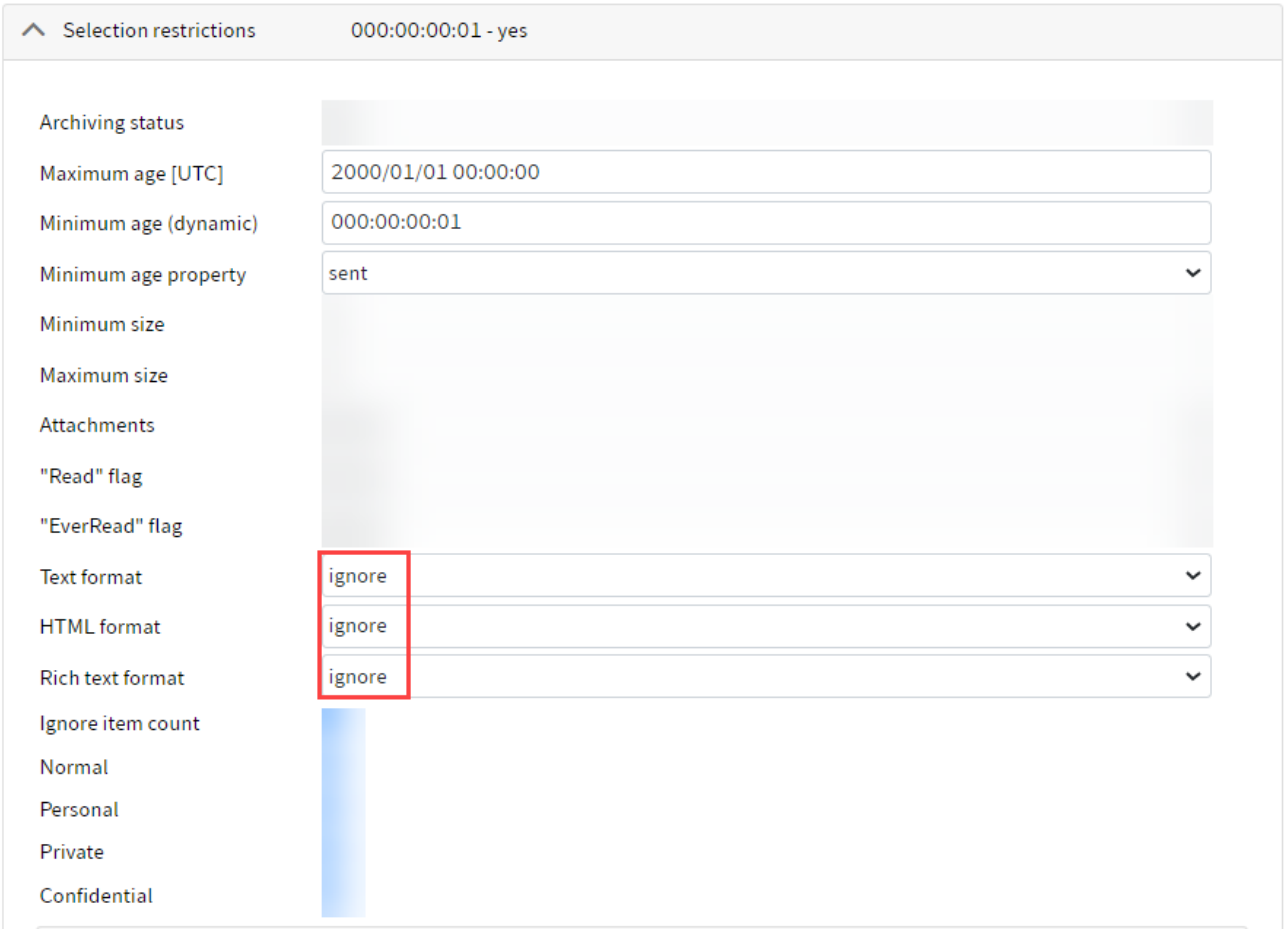


While you are creating the messages, you can still copy a short text into the message body and add text formatting (e.g. bold, italic, text color, text size). Of course, this does not work in text-only format.

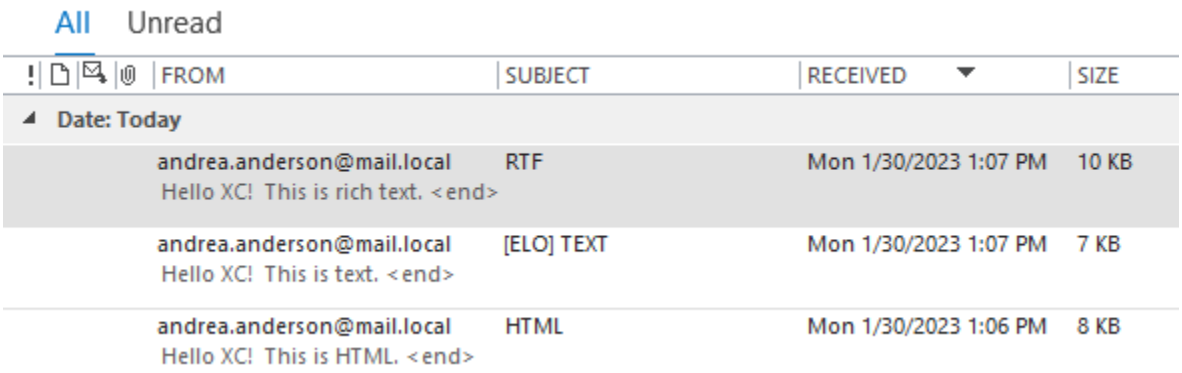
These messages are used for your next selection test:



Use the following selection restrictions:



Now select *yes* for text format and *no* for the other two formats. Test whether the correct message is processed:



If you use actions that read out the message body, the selection criteria for the body format can be relevant. The *Match/Replace* action in particular requires messages to have a specific body format if it is to correctly process them. The body is also just a message property whose value is the unformatted text content of the body.

Let us take a simple body, *Hello XC!*, which has been formatted a little for better visualization. The body stored internally is much bigger, although it is not shown in full here. We will only show you the HTML *body*:

```
<body
  lang="EN"
  link="#0563C1"
  vlink="#954F72"
  style="tab-interval:35.4pt;word-wrap:break-word"
>
  <div class="WordSection1">
    <p class="MsoNormal">
      <span style="color:#00B050">Hello </span>
      <b>
        <span style="color:black">XC</span>
      </b>
      <span style="color:black">!<o:p></o:p></span>
    </p>
  </div>
</body>
```

Based on this example, you can see that it is very important to know the body format in order to configure actions for match actions or even replace actions.

> **Information**
>
> HTML is the default message body format in Outlook. You can adjust this setting in the Microsoft Outlook options. We do not recommend using Rich Test Format (RTF). HTML provides the same formatting options as RTF and is much more widespread. If you use RTF, Outlook always attempts to store a version of the message body as HTML. However, this may result in server-side latencies.

Before you continue with the next example, set all selection restrictions for the message body back to *ignore*.

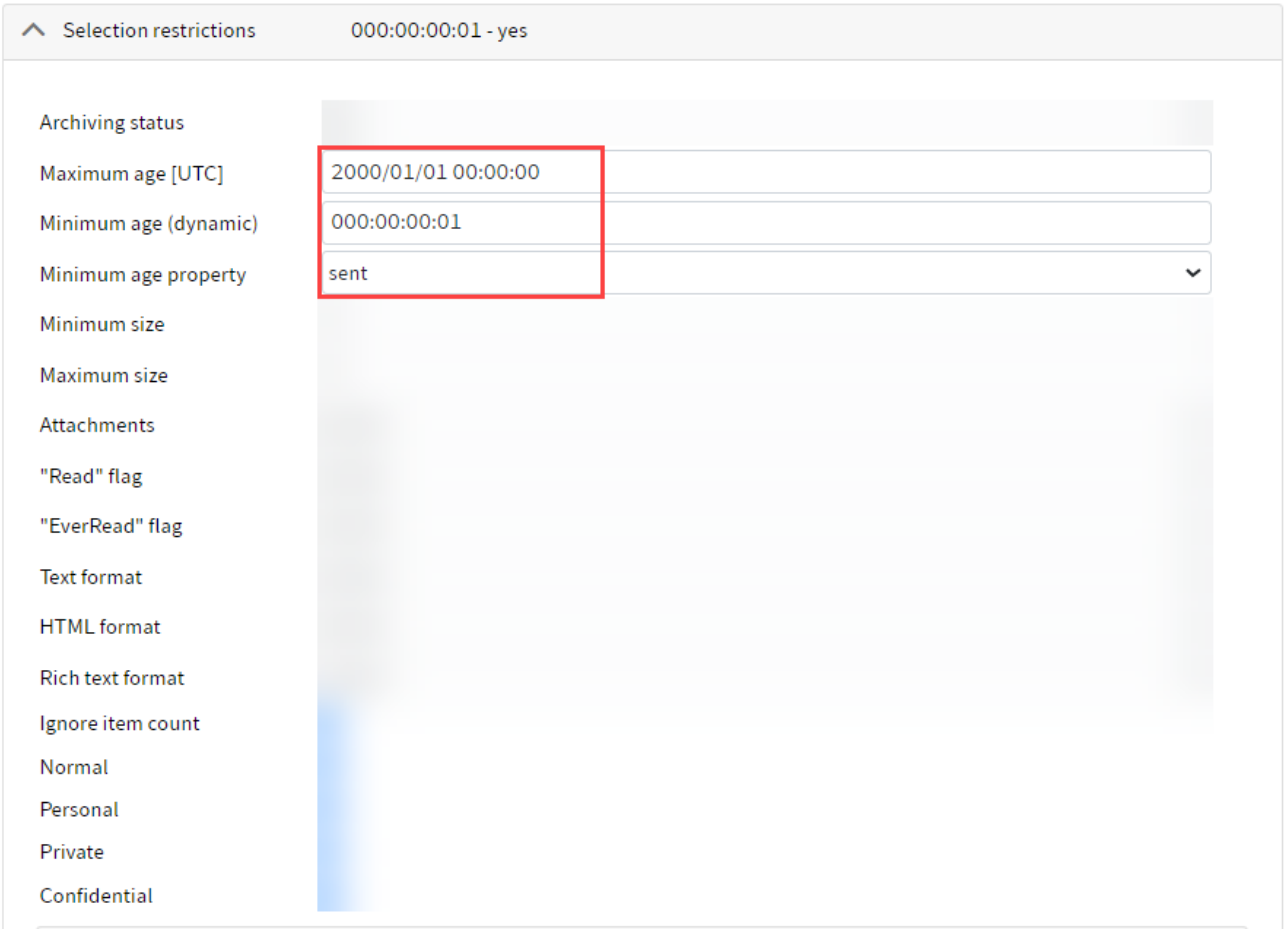
Message age

We haven't shown the example for setting the selection restrictions by item age until now despite the fact that time constraints for selected items are the most important criterion. However, to do this example, we need to consider another message class and adapt our tests somewhat.

The scenario is more complex and requires more preparation and exact configuration steps. Before reading on, create a simple e-mail with the subject *Old*. We will need it soon.

If something goes wrong while you are testing the selection restrictions by message age, start over.

We'll now explain how to use these selection criteria:

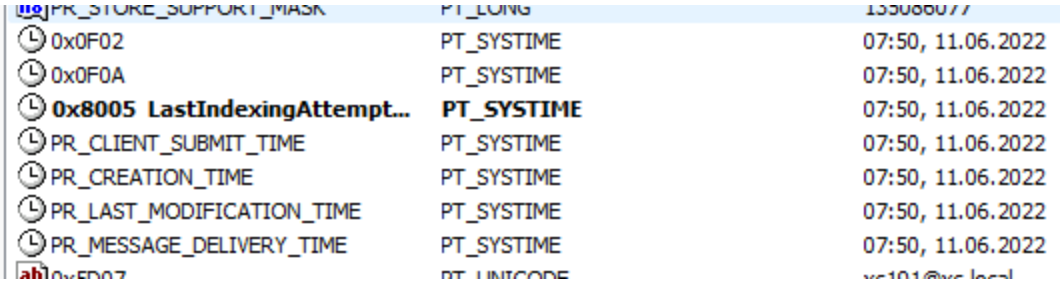


The default maximum age value is a static setting. Given the current year, you are unlikely to have any e-mails that are over 20 years old. However, you can change this value if this is the case. The theoretical limit is 01.01.1970. If you change the maximum age for any reason, make sure you retain the correct format.

The parameter format is also important for the minimum age selection criterion. It corresponds to the standard time format in ELO XC, i.e. days followed by hours, minutes, and seconds. As there is no default value for the minimum age, it is not automatically set when new action trees are created and has to be defined manually. If the *minimum age* is missing, the configuration cannot be validated.

The calculation of the message age requires a reference time, which is set by ELO XC to the time of the job start. This creates a similar message age for all messages selected in a job by any number of action trees. This internal timestamp of the job start is also used when setting the archiving status. If a message is assigned the *archived* status, the hidden property *EloXcArchivedBase* is given exactly this timestamp.

The third minimum age property selection criterion determines which value should be used to determine the age. Theoretically, you could use different time properties.

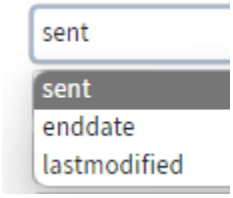


The figure shows which time properties are used internally by Exchange or Outlook. They serve different purposes. However, each gets the *PidTagMessageDeliveryTime* property (or *PR*MESSAGE*DELIVERY_TIME*). This timestamp is created at the moment an item is delivered to the inbox and is considered *received*. ELO XC uses the alias *sent* for this property.

PidTagLastModificationTime (or *PR*LAST*MODIFICATION_TIME*) can also be relevant for processing in ELO XC if you only want to select messages that the mailbox owner has not changed for a specified duration.

The *PidTagEndDate* time property (or *PR*END*DATE*) is not visible in the image. This is because this property is only created for calendar items that belong to a separate message class. The image above was created for an item of the *IPM.Note* class and therefore cannot contain *PidTagEndDate*.

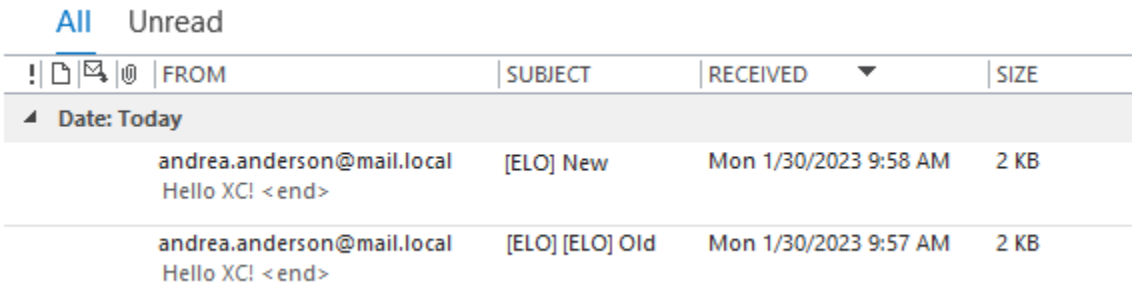
You can configure these three properties to determine the message age (general item age):



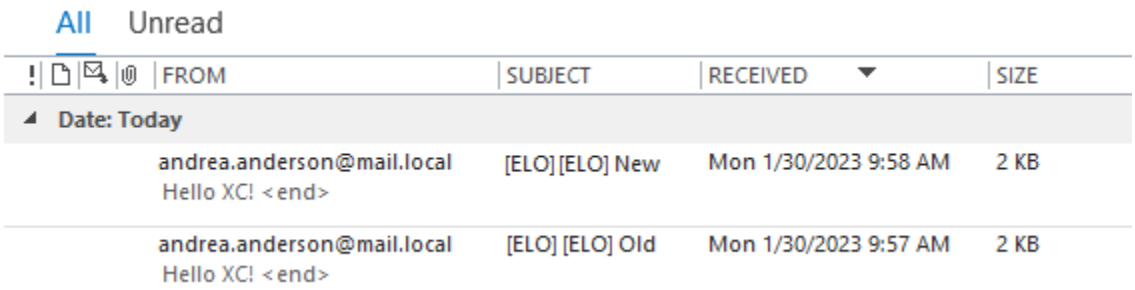
Parameter	Property
sent	PidTagMessageDeliveryTime
enddate	PidTagEndDate
lastmodified	PidTagLastModificationTime

If more than five minutes have passed since the *Old* message was created, you can create another message with the subject *New*. Afterwards, change the *Minimum age* restriction to `000:00:05:00` and check whether the correct message is processed:

Now set the minimum age to `000:00:00:01` again and check whether the correct messages are processed:

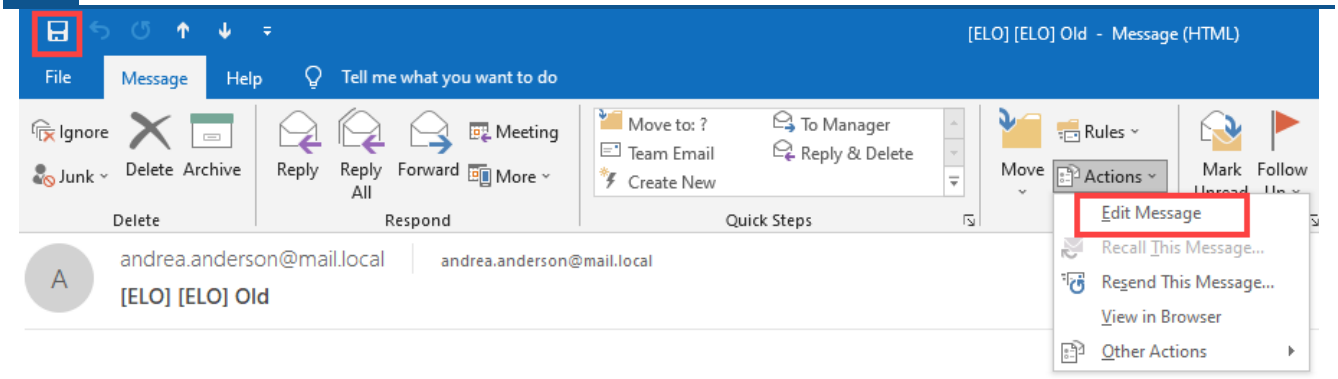


We now want to leave the minimum age at `000:00:00:01`, but still exclude *Old*. To do this, we set the maximum age to `2022/06/11 08:00:00` (note UTC conversion). This means it is newer than the *Old* e-mail. The result is:

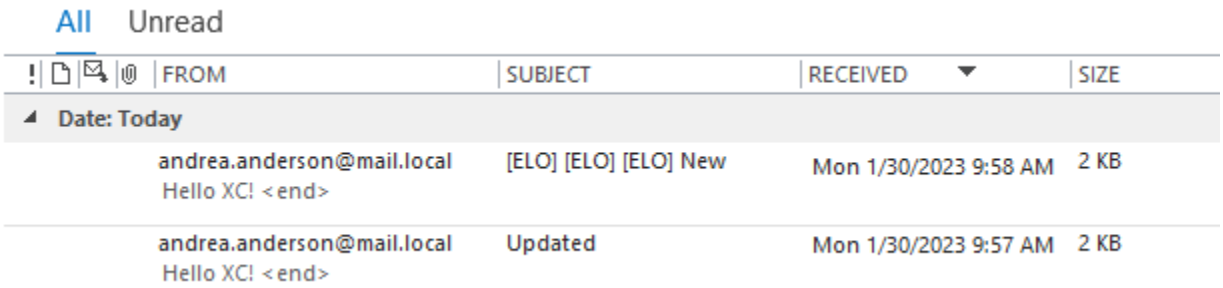


Now we change the matching property to *PidTagLastModificationTime* by selecting the *lastmodified* option. The minimum age should be set to 30 seconds:

The important thing is that we can get the configuration ready and start processing immediately. But first we need to change one of the two messages. Double-click the message to open and edit it and change the subject from *Old* to *Updated*.

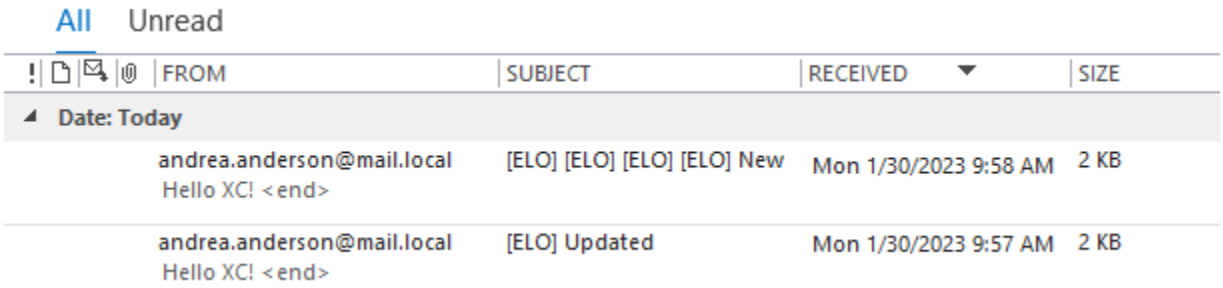


You can click on the subject line and just type *Updated*. You can take your time with this because Exchange doesn't know about this change yet. After you save the change by clicking on the disk icon, you have 30 seconds to make the effect of your configuration change visible:

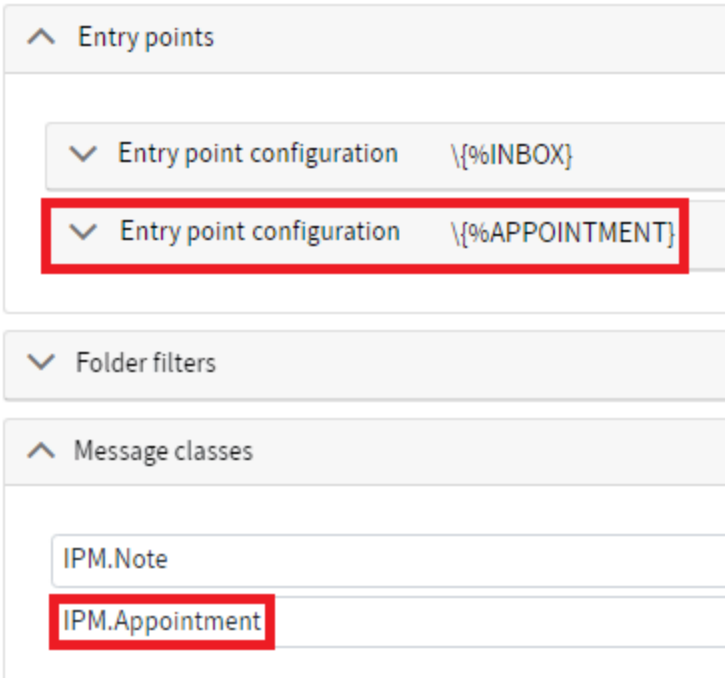


Updated, the former *Old* e-mail, was not archived, even though the message is significantly older than *New*. This is exactly what we want to achieve. The minimum age of 30 seconds since the last change has not yet passed, which is why ELO XC did not select this message for processing.

Now change the minimum age back to one second and set *sent* as the selection property. To be sure, start processing again so that you get this result:

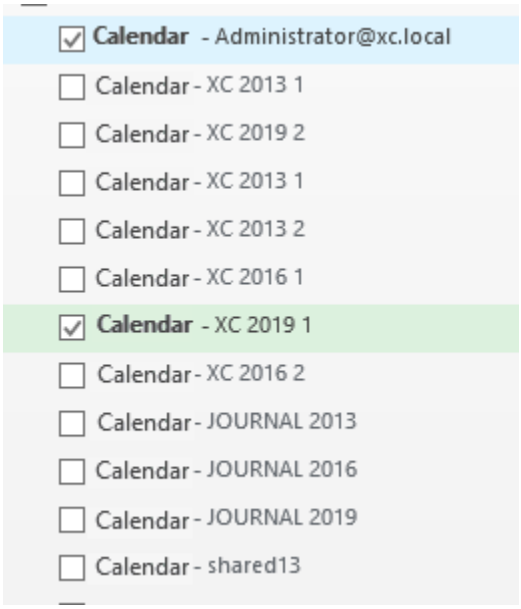


Now we can look at the *enddate* (*PidTagEndDate*) selection. First, however, we need to give access to calendar items (*IPM.Appointment*). In this case, we need to make changes to the entry points and allowed message classes of the action tree:

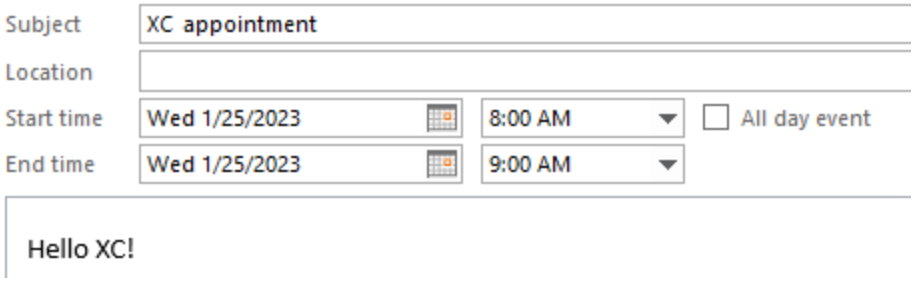


Configure the instance to start immediately and clear the inbox. Now create a message with the subject *No appointment*.

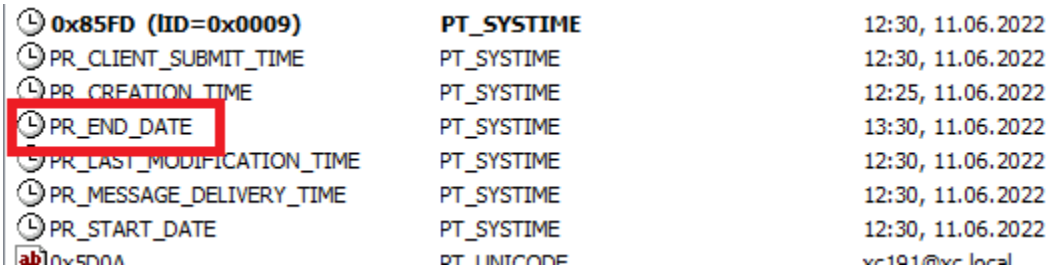
Before you create the appointment, which we also need for the next example, you may need to take into account that when you click on the calendar, Outlook opens the associated folder of the main mailbox in the Outlook profile. If you follow the examples with another mailbox associated with the Outlook profile, you have to view this calendar first before you can create the appointment:



If you are doing all the examples with only one mailbox anyway, and that mailbox is used to send all the test messages, you can ignore this interim step. Now create an appointment in the calendar and set it to end 30 minutes into the future:



If Outlook issues a warning that the appointment has already started, you can close the message. It is not relevant for this example. Before the 30 minutes until the end of the appointment have elapsed, we run the next test. Change the matching property to *PidTagEndDate*, which only uses *IPM.Appointment*:



You need to change the required minimum age of the selection restriction:

Process the e-mails again after this change. You will notice that neither the *No appointment* e-mail nor the *XC appointment* e-mail are processed.

This is because the end time of the appointment is in the future. This means that the calculated minimum age cannot have been reached yet. In addition, standard e-mails (*class IPM.Note*) do not have this property. The result is that neither the e-mail nor the appointment can be processed. This may seem like a disadvantage at first, but it is an important advantage when processing Outlook calendars.

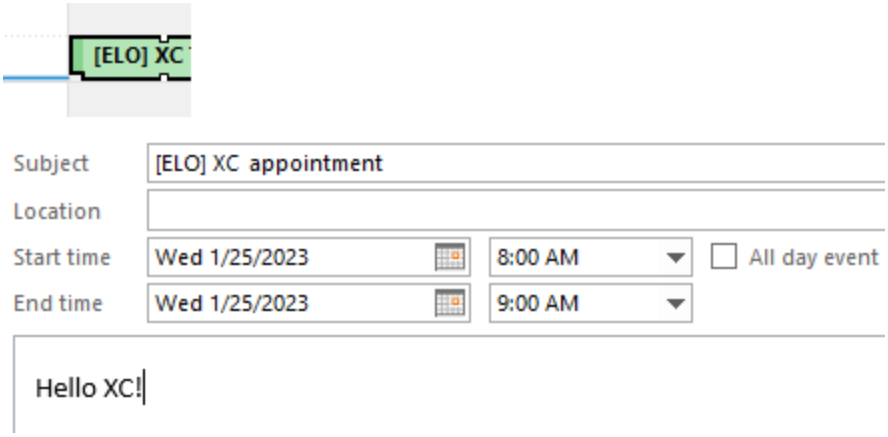
Imagine that ELO XC processes thousands of appointments using only the creation date as a selection criterion for processing. Does this mean that ELO XC should archive every appointment that is created today and does not take place until six months from now? It quite possibly makes sense. In this case, the creation date of the appointment or the arrival time of the appointment in the mailbox is sufficient.

However, if you assume that ELO XC should not only store items, but also delete them, an essential basic function for reducing the amount of items stored on the Exchange server, then the risk of

losing data would be very high. Moreover, appointments are often scheduled far into the future. So how what unit should we use for the minimum age of the creation date: days, weeks, months, or years?

The best way to correctly process appointments without deleting the items from the mailboxes too early is to wait until the appointments are over and then delete them. In most cases, it is sufficient to archive the calendar entries immediately with a designated action tree and then delete them right away. In principle, however, you still have the option to select different reference points.

Process the e-mails again after the end of the appointment entry. You will now see the expected result in the calendar:



Information

If you configure ELO XC to process e-mails and calendar entries, it makes sense to process the two message classes separately. Since no e-mails are selected with *enddate*, you can safely assume that there will be no unwanted processing results, but this does not apply vice versa, nor does it make sense to use complex, overlapping selections. This could lead configuration errors and unwanted results in the long term. A higher number of action trees can noticeably increase the time it takes a job to run, but it is not always necessary to process items faster, especially since there are alternative measures to increase performance in ELO XC.

Reset all selection parameters to the default values (age: 1 second, *sent* property) and remove the message class *IPM.Appointment* as well as the entry point *{%APPOINTMENT}* if you want to do the examples in this section again.

Best practice

In practice, selection restrictions are used to ensure that ELO XC processes the right message set and that the selection criteria are as ideal as possible. The processing time differs if only those messages that an action tree is actually going to process have to be fully retrieved from Exchange. In contrast, restrictions with low selectivity mean that messages that have already been (down)loaded must be discarded again by performing subsequent property matches. Using accurate selection criteria ensures that the Exchange server makes the selection before the

messages are fully loaded and only the IDs of the relevant messages are determined. Complete item data is retrieved based on this ID only when processing starts.

Exchange server performance also depends on the total number of mailbox items being managed. Having a large number of items can cause delays. You should always configure ELO XC in more complex scenarios in such a way that the items are deleted from the Exchange server as early as possible. If an instance has two action trees, one for filing and one for deleting old mailbox items that have already been filed, we recommend that you run the action tree for deleting first. This reduces the total amount before the messages are selected, which improves performance.

This improvement is already incorporated when selecting the mailbox folders to be processed by generally excluding folders without items from the item selection, as the item count is generally available when querying available folders. Only when processing *public folders* do we recommend processing folders independent of the determined item count (see selection option Ignore item count), as the system may return incorrect item counts for *public folders* when queried and and so these folders must be subjected to an item search in each case.

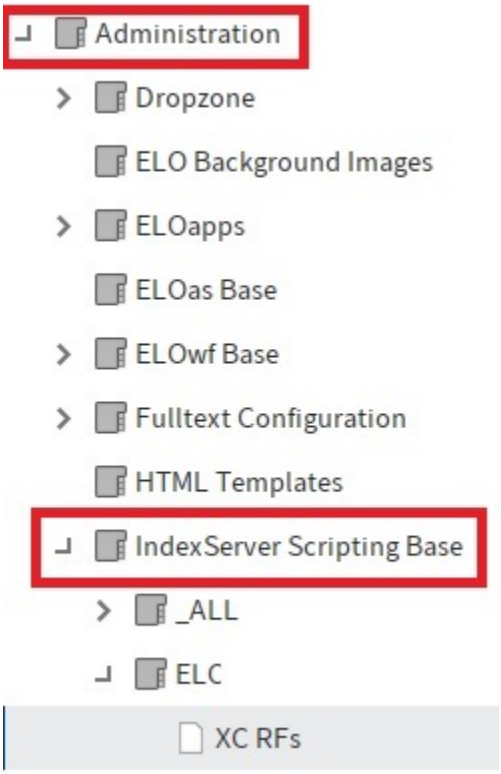
You can also use the *Move* action to reduce the total number of messages in advance. If mailboxes contain messages that should never be processed, it is worth considering moving them to a child folder that is ignored during message selection. However, if mailbox owners want or need to decide which messages will be processed, you could specify a child folder that will be processed on its own. You can easily do this by configuring a suitable entry point. Finally, the *Tag* action allows you to assign an internal tag to messages to ensure that ELO XC ignores them on a permanent basis.

External calls

The *External call* action is a good way to trigger next work steps in ELO after ELO XC has stored the message. Here are some examples of how you can configure this in ELO XC.

Registered functions

Registered functions are JavaScript functions that are directly controlled by ELO XC via the Indexserver interface. They are stored in the Administration folder of the repository:



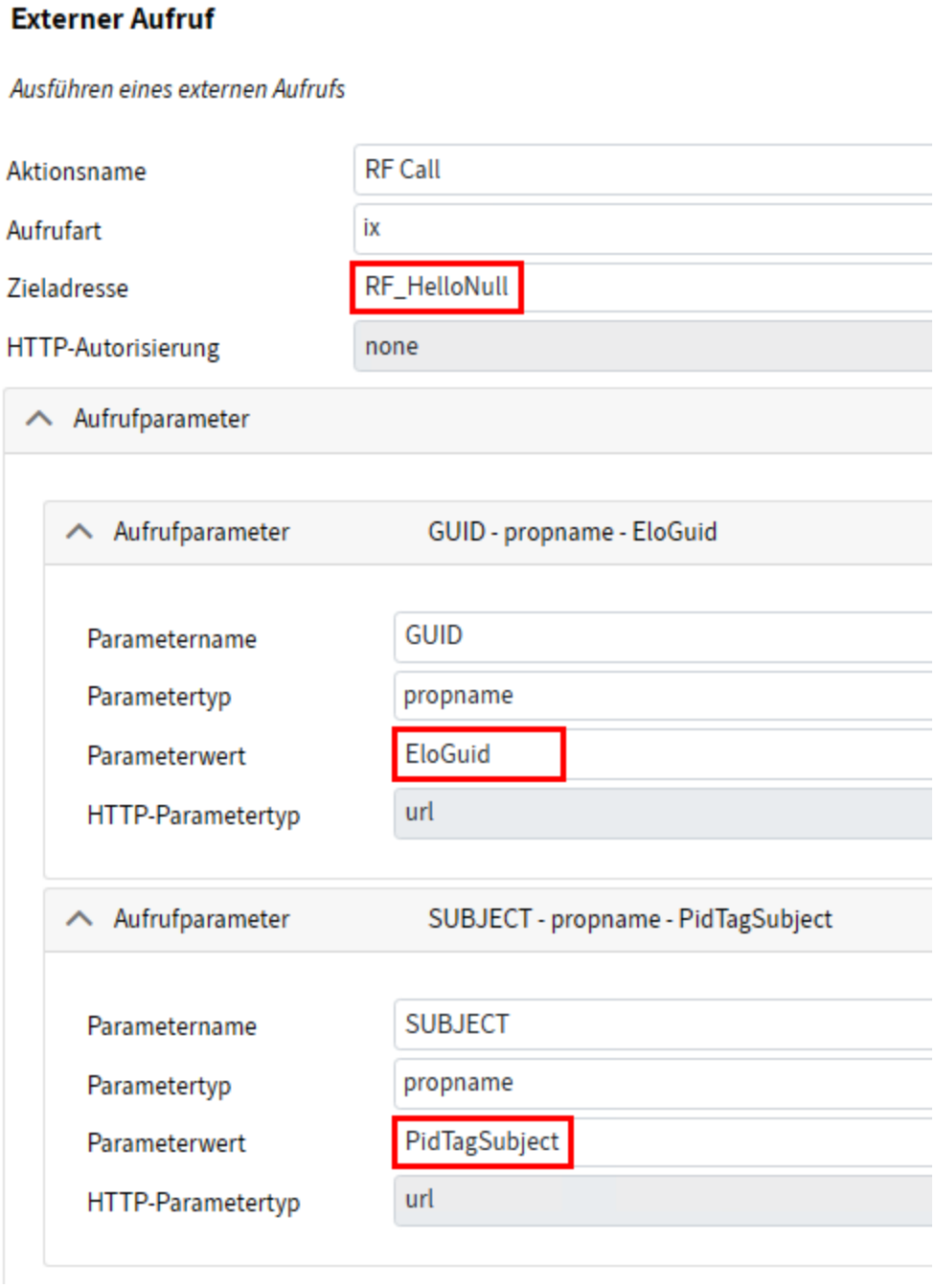
In this example, the script file *XC RFs.js* is generated and stored in the repository. The script functions demonstrate how to handle function parameters and function returns:

The *RF*HelloNull* and *RF*HelloNullReturn* functions return the same null result. *RF*Hello*, on the other hand, returns `Hello`, while *RF*HelloArgs* attaches the first function parameter to *Hello* and creates `Hello Arg 0: <parameter value>`.

For more information about using the Indexserver interface, see: https://docs.elo.com/dev/programming/en-us/elo-indexserver-programming-guide/

You will find examples of registered functions at: https://docs.elo.com/dev/programming/en-us/elo-indexserver-programming-guide/added-functionality-with-registered-functions/#example-of-a-registered-function-in-javascript

In order for ELO XC to call the correct function, the *External call* action must be configured appropriately:



The name of the registered function has to be used as the target address. The parameters are always passed in the order of the configured call parameters. On transfer/call, a parameter field contains the value determined at runtime based on the configuration.

The example configures the call for the first example function. In the worker log, you will therefore find the result `null`. If XC calls `RF_Hello`, `Hello` appears in the log.

The example *RF*HelloArgs* should show the SORD GUID of the message filed previously in the log. If you change *RF*HelloArgs* to use `args[1]`, you will get the subject of the message in the log instead of the GUID.

Please note

Registered Indexserver functions are the most flexible and comprehensive way to do post-processing after storing messages, and should therefore always be considered in such cases. However, care should be taken to ensure that the processing time of these functions is kept to a minimum. ELO XC processes masses of e-mails, which means that latencies of just a few seconds have a significant impact on the overall processing performance. Therefore, functions that quickly categorize a message and forward it to appropriate scripts or components in ELO are recommended.

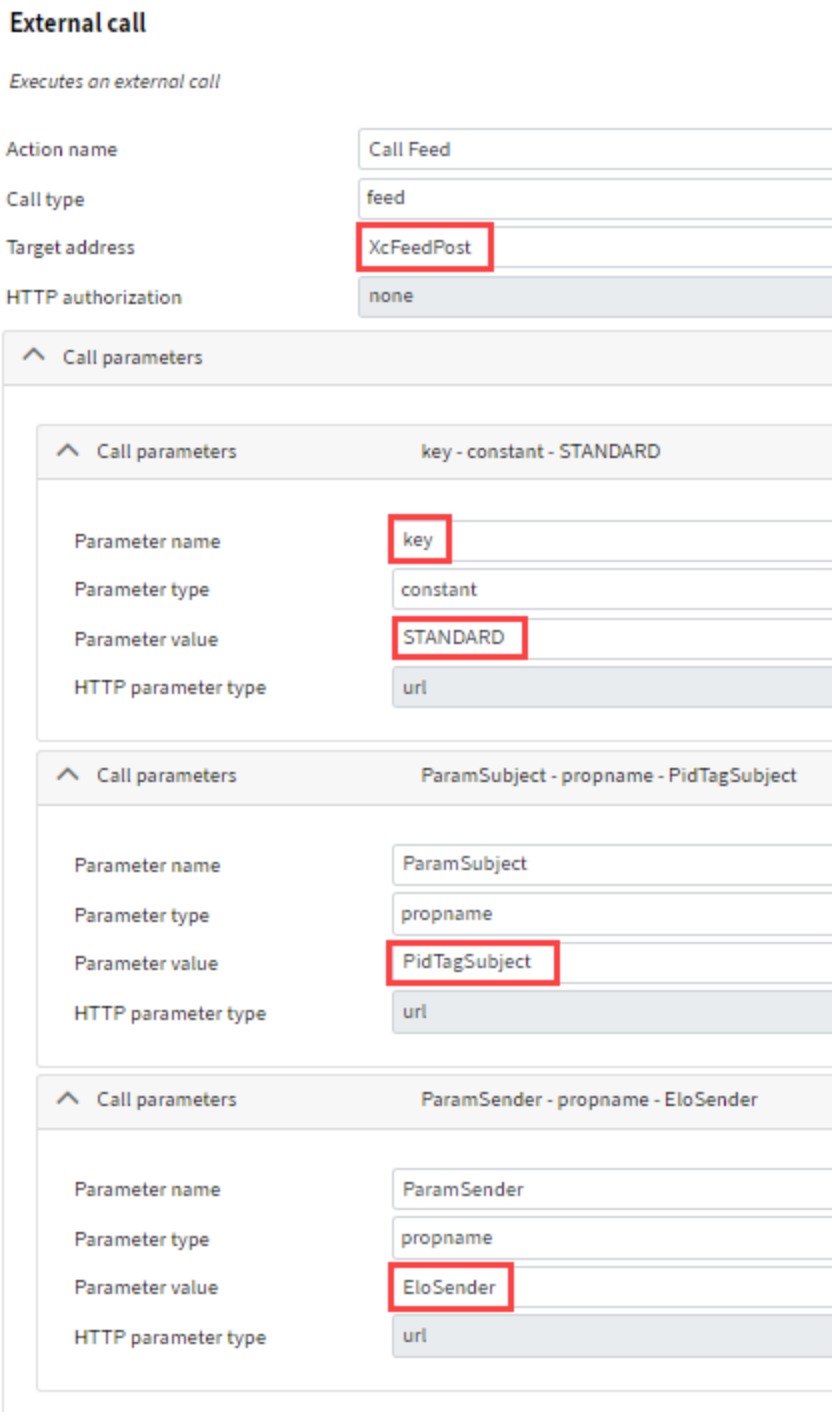
ELO feeds

The *External call* action can also be used to generate ELO feeds. The following example shows how a localized action text can be generated in the feed after storing an e-mail. This can be used to notify users about incoming messages, for example.

The localized texts must be available as properties files in the Administration folder of the repository:

They define the localized action text to be displayed. The freely selectable variables/parameters of the message are listed numbered in curly brackets.

The action configuration looks like this:



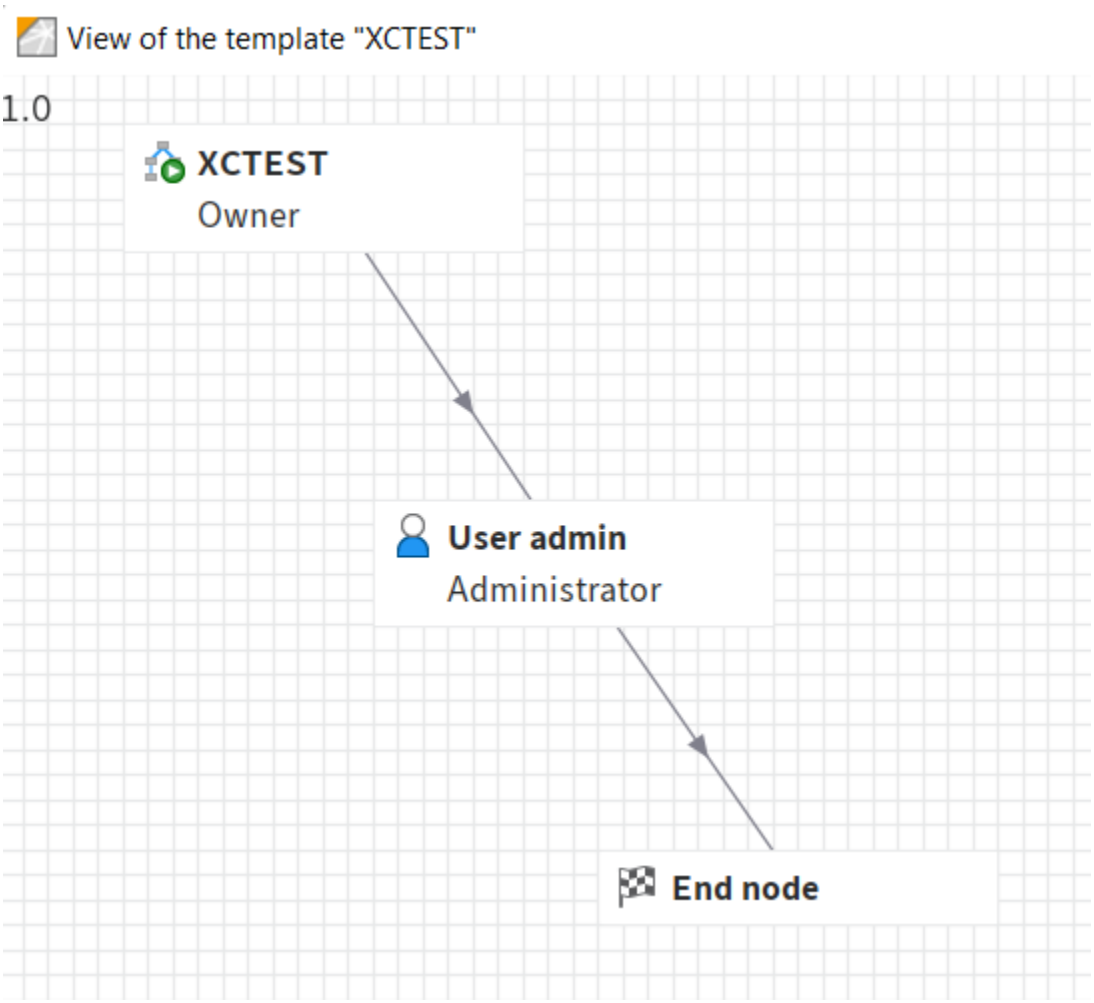
The name of the properties file is the call target.

The STANDARD text value named key has to be entered as the first call parameter. This entry is mandatory in order to determine the correct entry in the properties file.

The other call parameters are determined at runtime and transferred in the order they are configured to generate the feed action. The order of the call parameters should correspond to the numbering of the text variables.

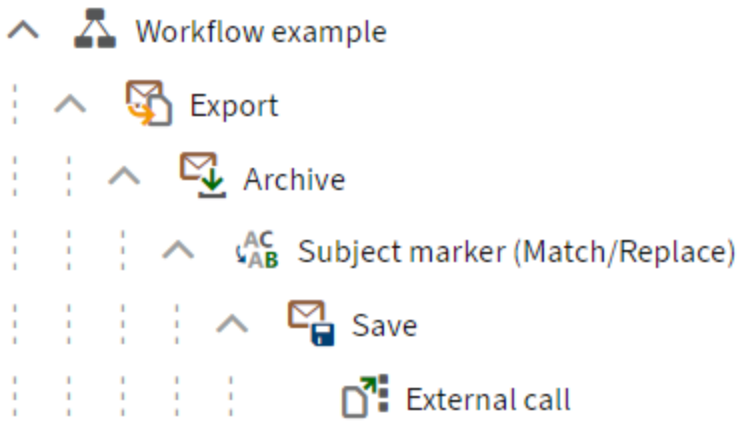
Workflows

To pass a message or attachment to a workflow using an external call, you need to first create a workflow template. To illustrate the action, a simple template is enough. We'll call it *XCTEST*:



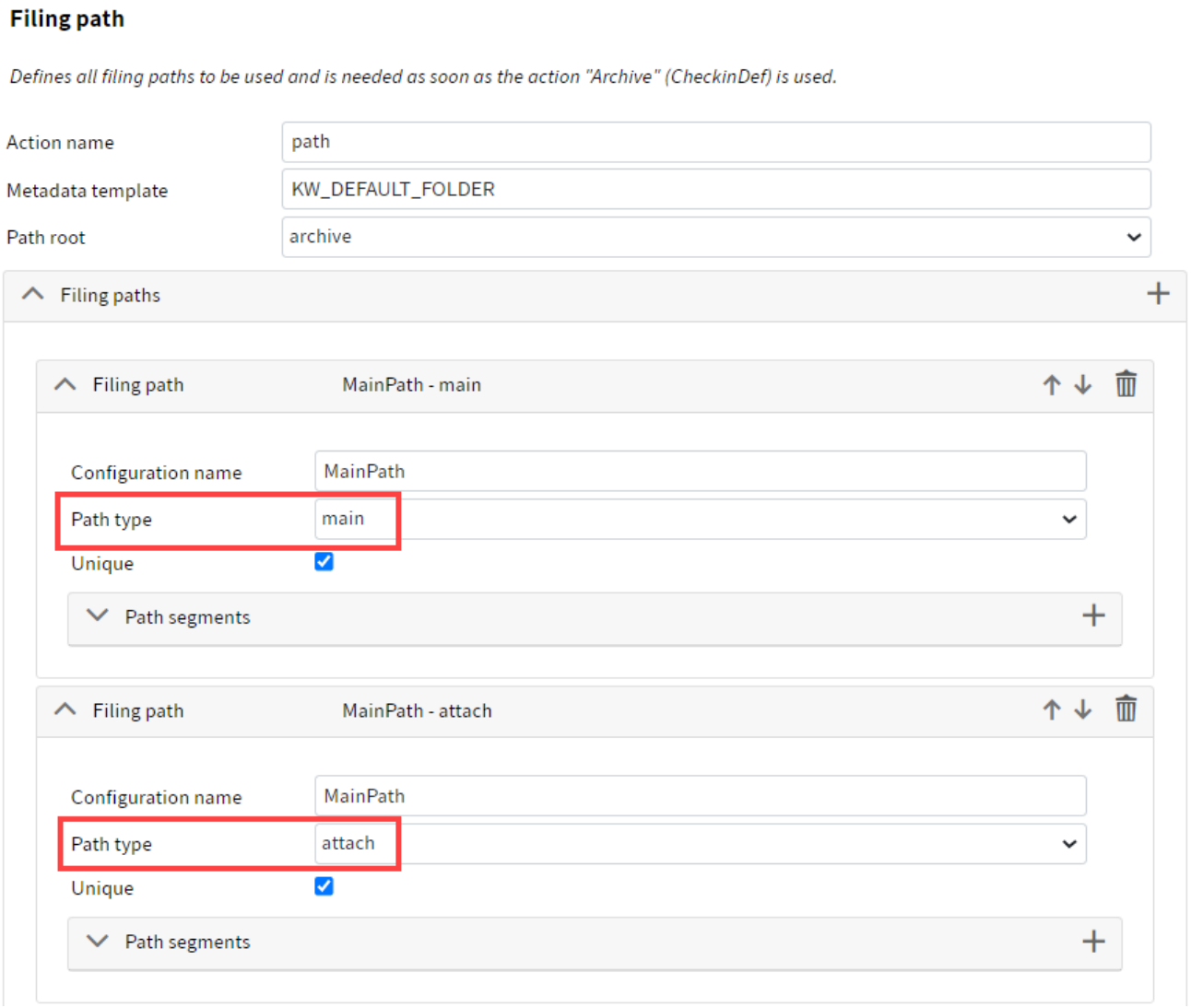
The name of the workflow template is used in the ELO XC configuration to create a new workflow that is used while processing the current message.

Configure a simple action tree with the actions *Filing path*, *Export*, *Archive*, and *External call*.



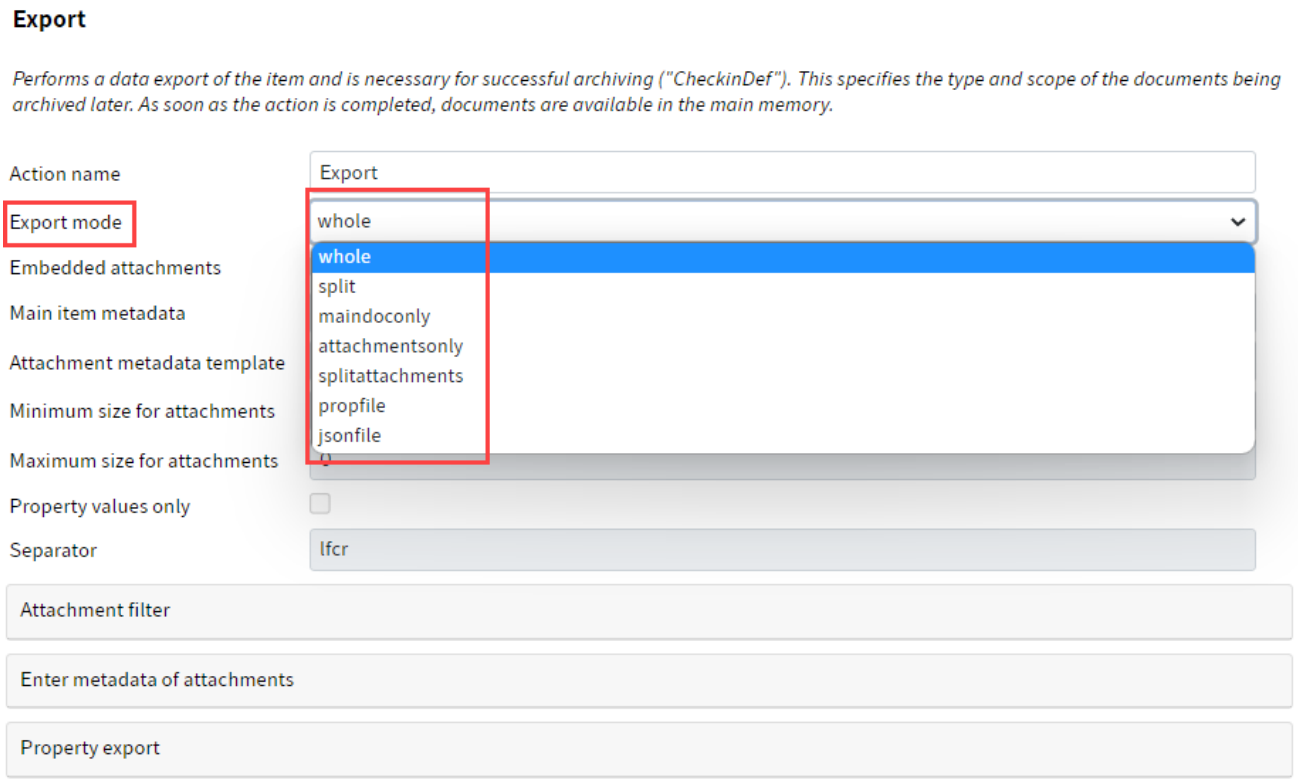
The tree selects non-archived messages, uses the Inbox as the entry point, and only processes the *IPM.Note* message class. The *Archive*, *Match/Replace* (subject marker in this case), and *Save* actions can be used with the general settings or changed if required. The latter two actions are not important for calling the workflow and are used here to make it easier to execute multiple tests one after the other while processing only current example messages.

In order to use the filing path for messages (main documents) and attachments in different export scenarios, you must configure a path for both document types.



The path segments are not important in this example. However, they were chosen so that it is easier to recognize which documents are stored in the repository while testing different configurations. These segments are configured for both path types:

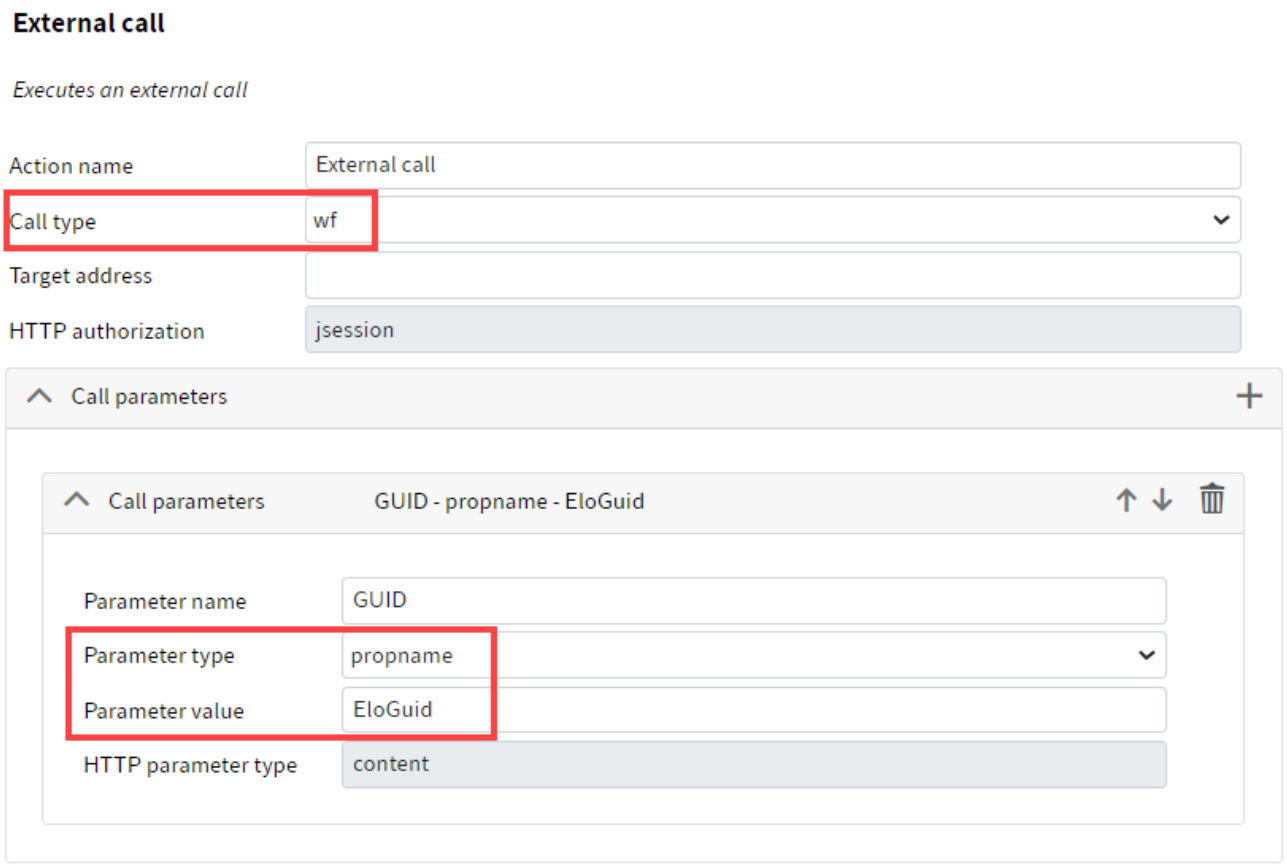
In the *Export* action, it is evident that both file types are required. The export mode automatically determines which path type should or can be used.



Depending on the export mode, ELO XC tries to determine/create the correct filing path while the *Archive* action is processing. The table shows how the correct path is determined.

Export mode	**Path determination**
whole	Only the configured main path is used.
maindoconly	Only the configured main path is used.
attachmentsonly	Only the configured attachment path is used.
split	The configured main path is used for the main document, and the configured attachment path is used for attachments. If an attachment path was not configured, the configured main path is used.
splitattachments	The configured main path is used for the main document, and the configured attachment path is used for attachments. If an attachment path was not configured, the configured main path is used.

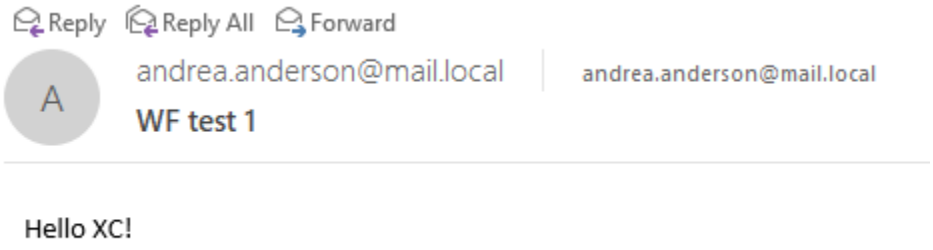
The configuration of *External call* must take into account that the call target is a workflow and that an existing SORD must be passed to the workflow.



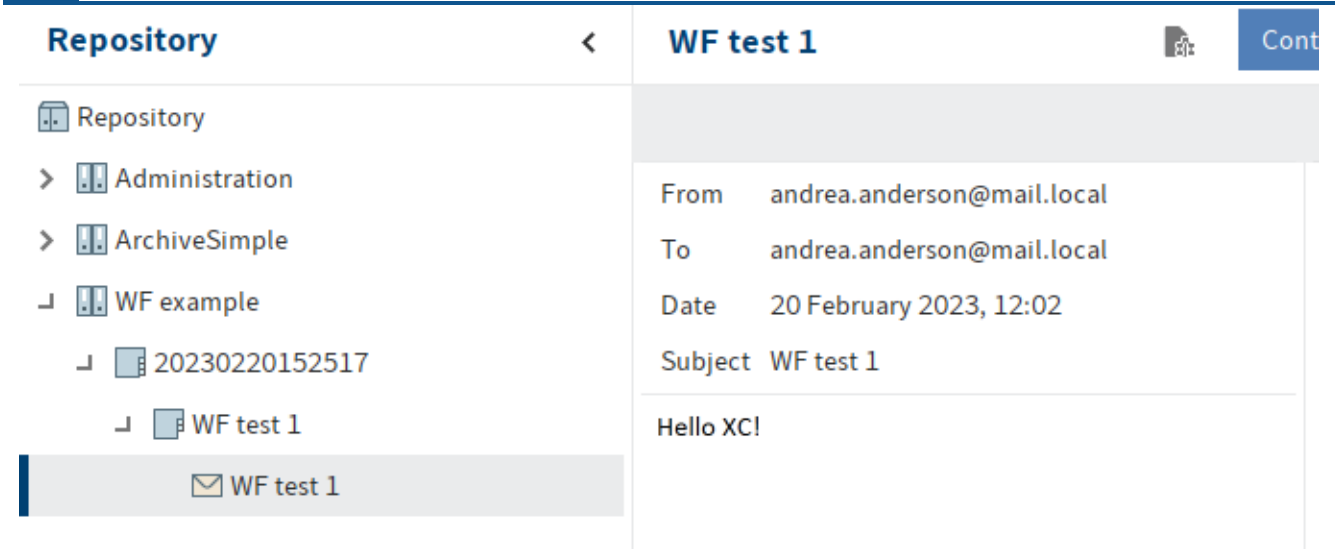
Since the action can call different interfaces, the *wf* call type must be specified for the creation of workflows. The GUID created in the process identifies the newly created SORD (document).

The *Parameter name* is not required for a *wf* call. You can define any value here. The GUID, on the other hand, can only be retrieved via the ELO property *EloGuid* – *propname* specifies that the configured parameter is interpreted as a property name.

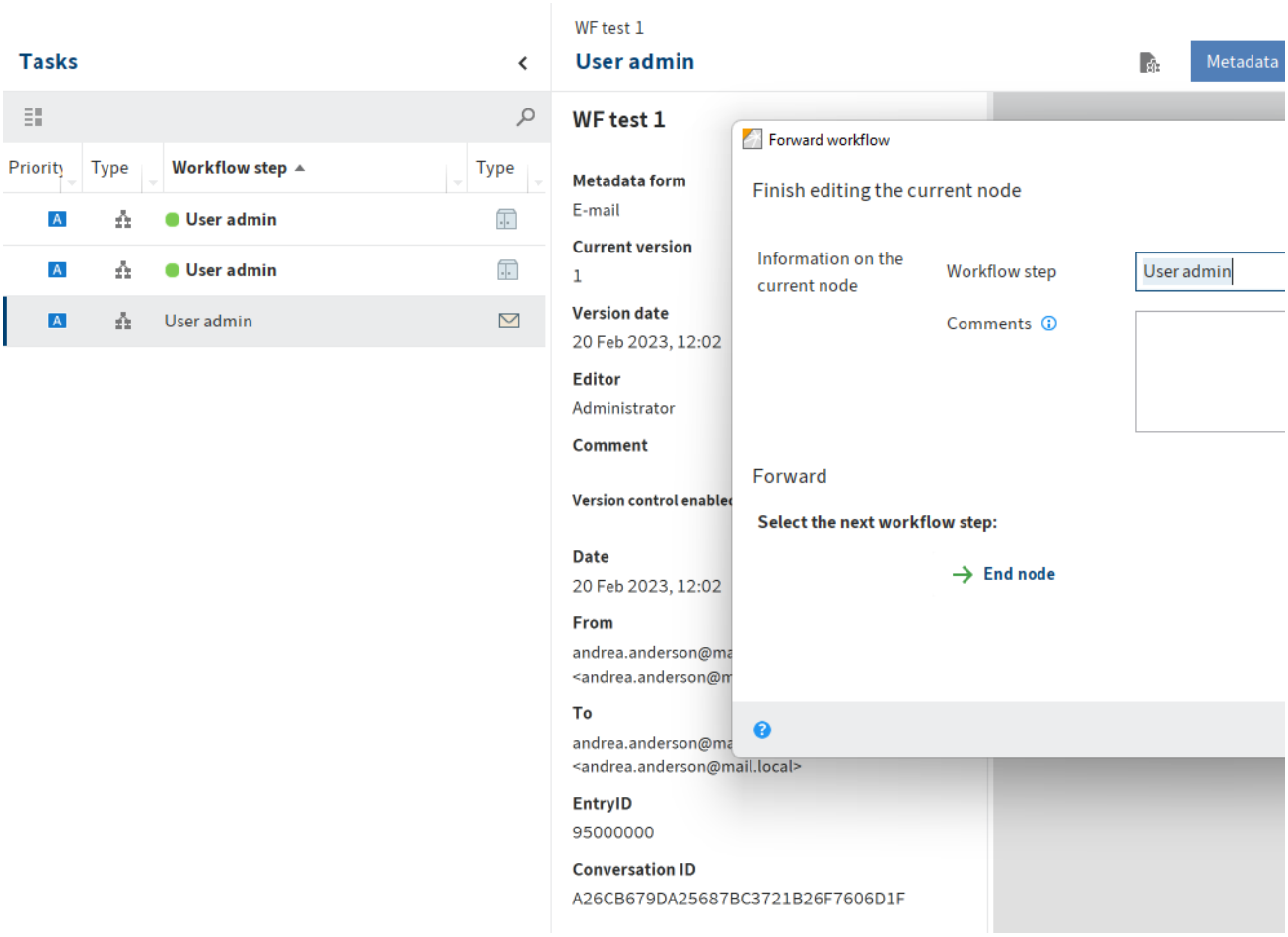
Test this simple configuration with a short message.



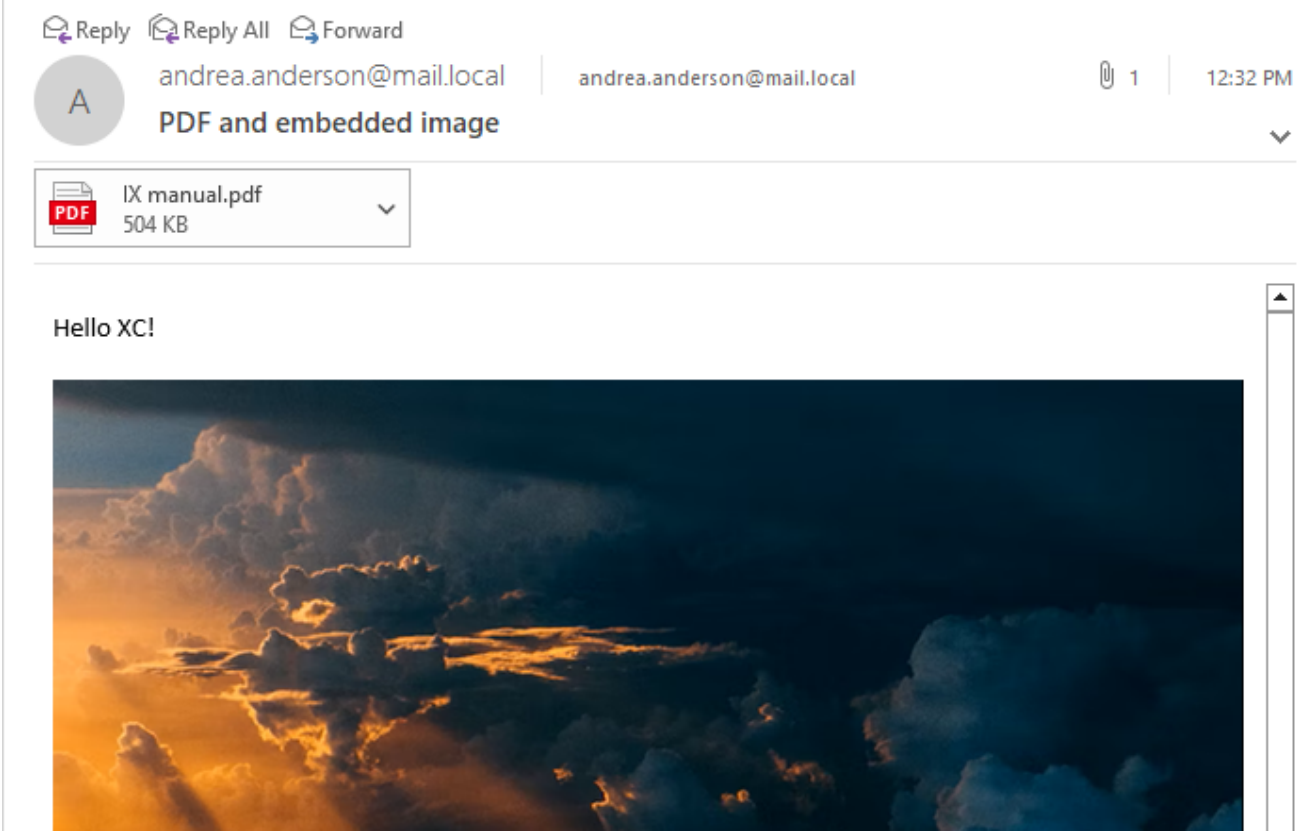
Publish the configuration and execute it once. This message should appear in the repository:



The user of the first workflow node (in this case *Administrator*) should also see this workflow as a task:



In the next example, store the message and an attached PDF file in separate folders in the repository. You only want to start the workflow on the PDF, however. Create a message with a PDF attachment and an image copied into the message body.



Which actions would the changes to the ELO XC configuration be made in?

- What?

  The number and type of stored documents are always configured with the *Export* action.

- How?

  The *Archive* action determines how the documents are stored.

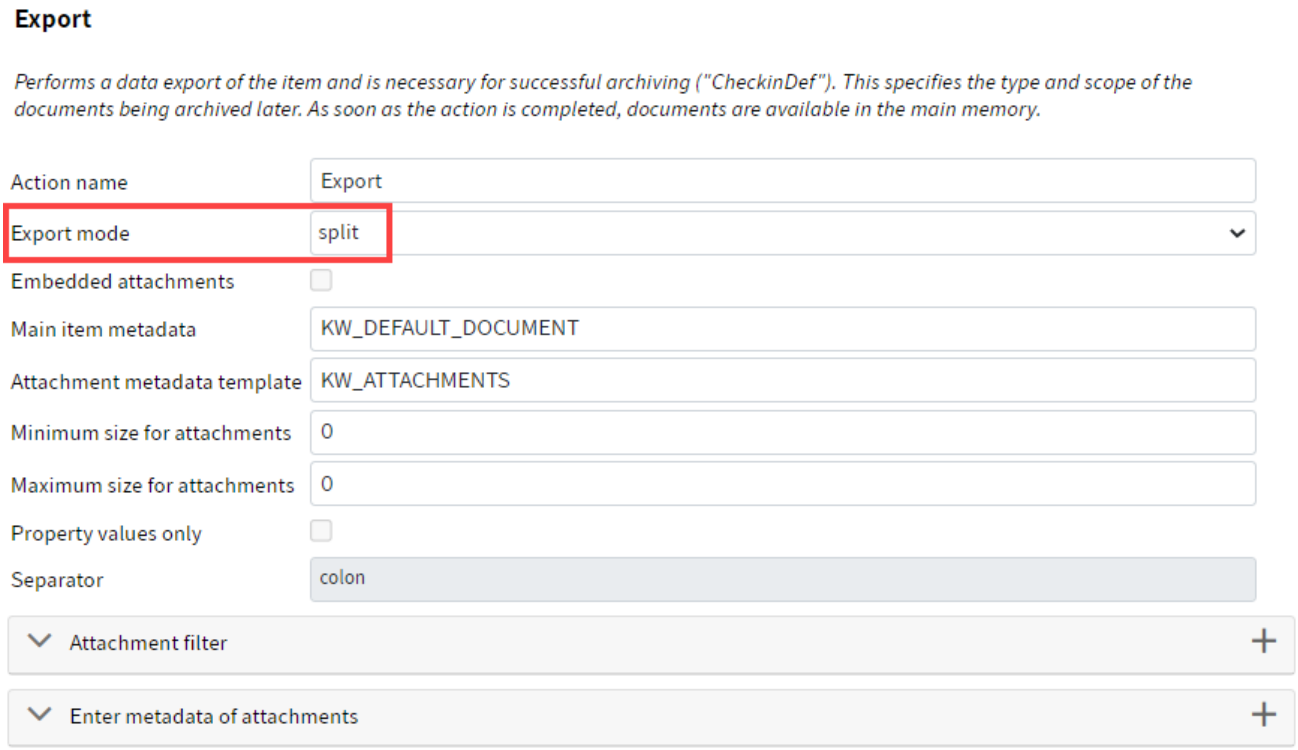
- Where?

  The *Filing path* action determines where the documents are stored.

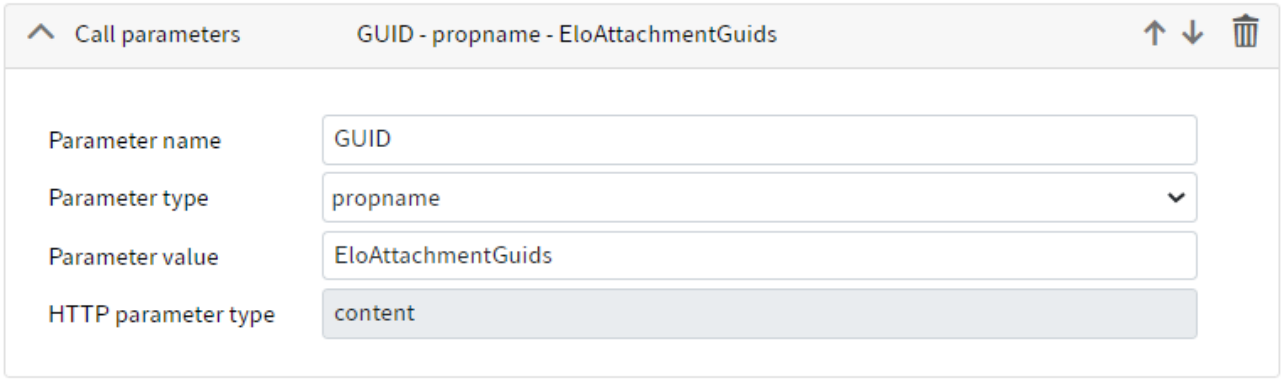
- And after that?

  You could continue processing the message with the *External call* action or simply delete it.

If you want to separate the PDF from the message, you need to configure the *Export* action. The *split* export mode extracts the attachment from the message. In this case, the message, or main document, is stored in the repository without attachments.

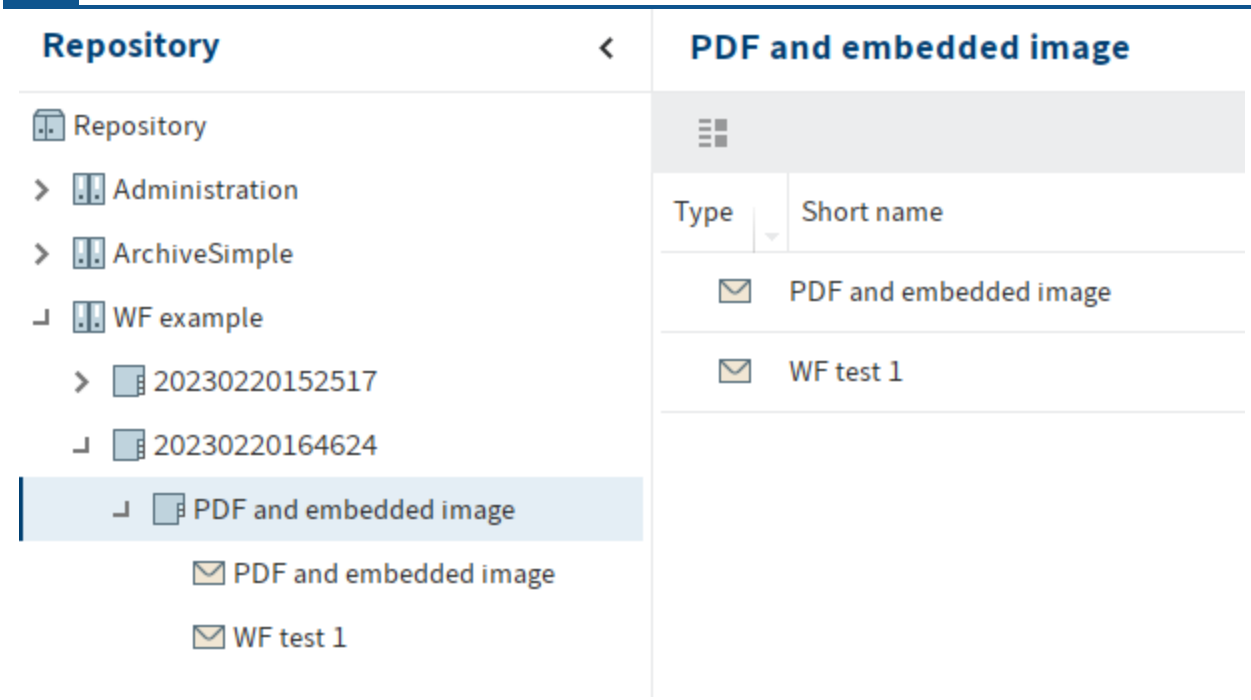


The second change you need to make in the configuration is less obvious. It is done in the *External call* action.

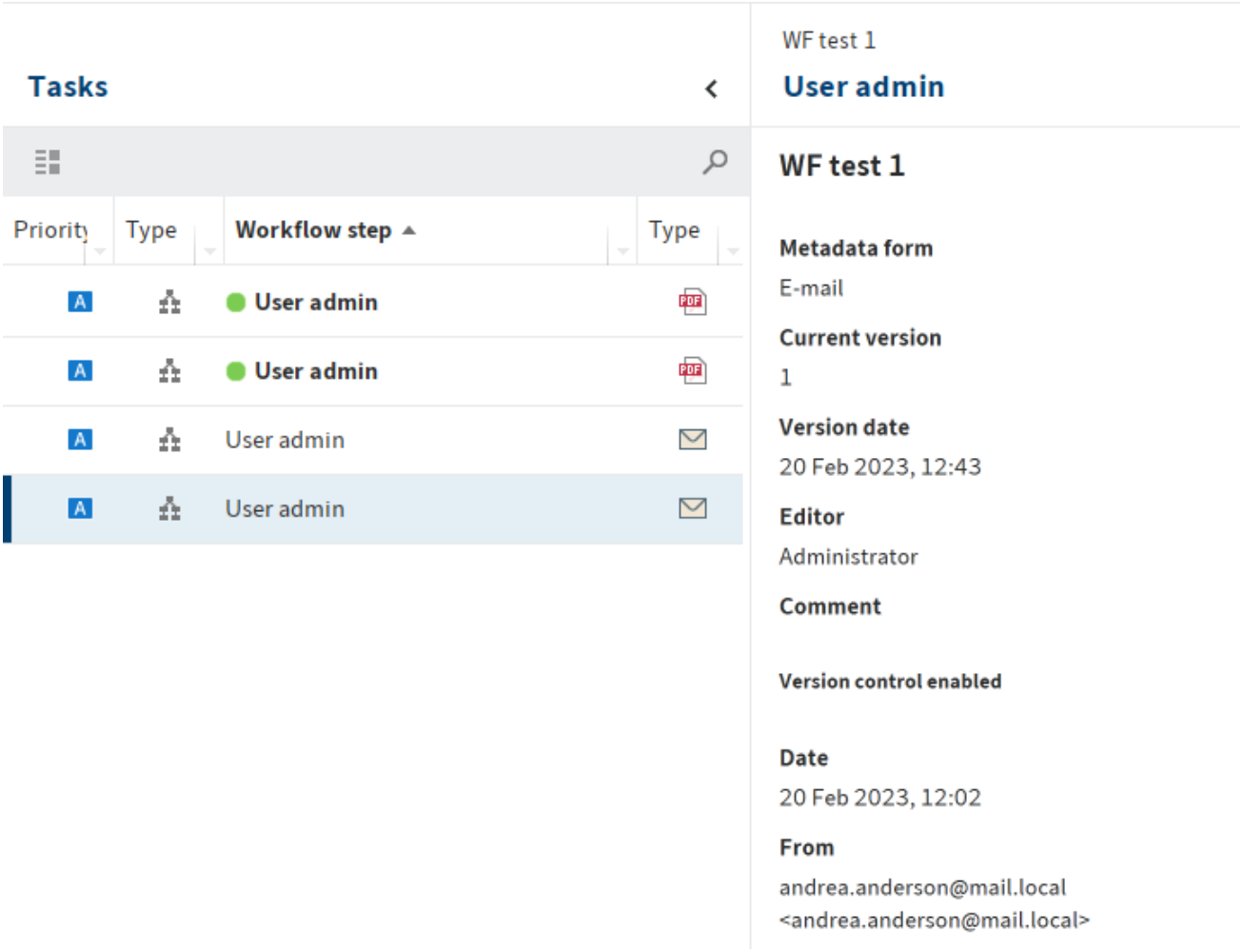


The *EloGuid* property can only be used for the main document, i.e. the message itself. If you want to access the GUIDs of extracted attachments, you must use the *EloAttachmentGuids* property. Note that an internal list of attachment GUIDs is always created since the number of attachments a message has can vary. How the list is subsequently processed depends on the context this list property was configured in. In an *External call* of type *wf*, the list remains internal. ELO XC uses it to start a workflow for each GUID in the list. If it is used in a metadata template, the GUIDs are applied as field values separated by commas.

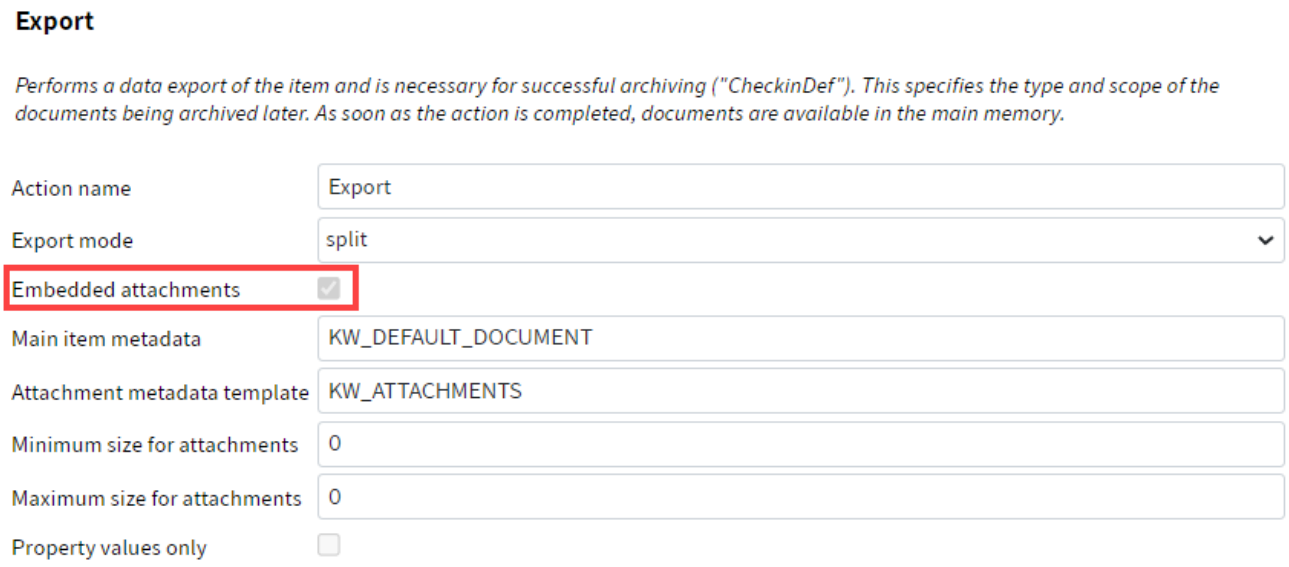
Publish your changes and start processing. You should now see the following result:



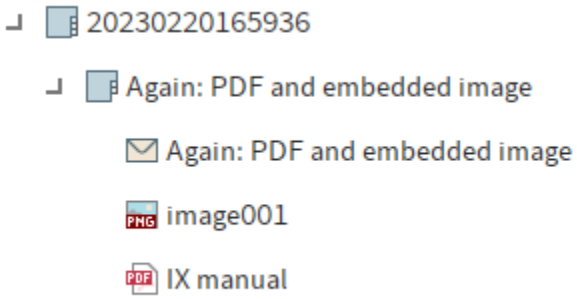
The new workflow is in the task list:



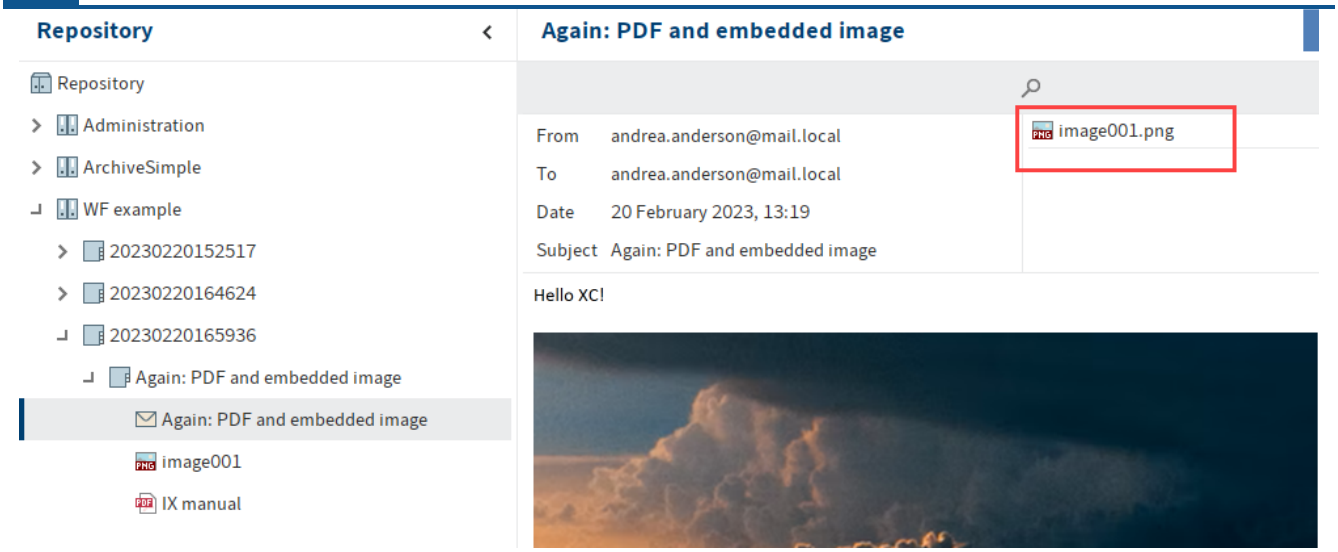
The embedded image is still missing. It is definitely an attachment, but it hasn't been processed yet because it is not a *file attachment*, but an *inline attachment*. Change the *Export* action again by enabling the option to include inline attachments.



After processing, you will see all parts of the message in the repository:



You can also see that the extraction process did not work correctly since the file attachment is missing:



This is due to the nature of inline attachments. They are associated with the message body, which is an intrinsic part of the message. File attachments, on the other hand, are removed from the message. There are now two workflows for the configured export mode – one for the PDF and one for the embedded image.

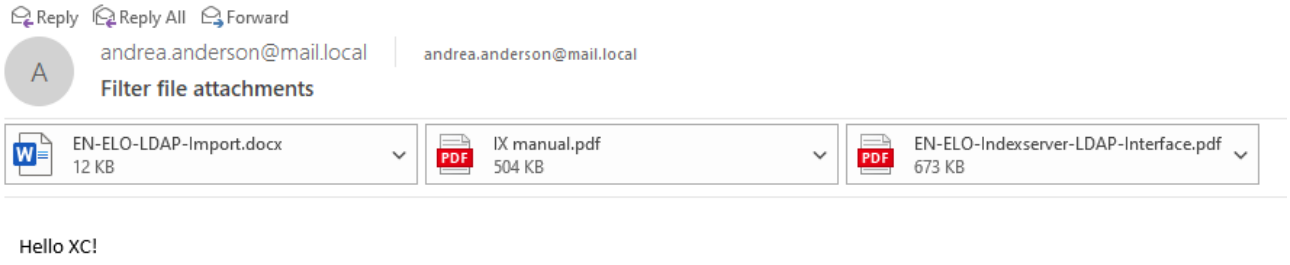
Along with *EloGuid* and *EloAttachmentGuids*, you can forward the folders of the stored documents to a workflow with the *External call* action. The properties used to identify SORDs in the repository are:

EloRecipientAddressesBcc	String	Specifies the complete BCC smtp list of an item
EloGuid	String	Specifies the ELO guid valid for respository items
EloParentId	String	Specifies the parent id of the main ELO respository item
EloAttachments	String	Specifies the attachments list of an item
EloAttachmentGuids	String	Specifies the guid list of all archived attachments
EloAttachmentParentIds	String	Specifies the parent id list of the attachment ELO respository i

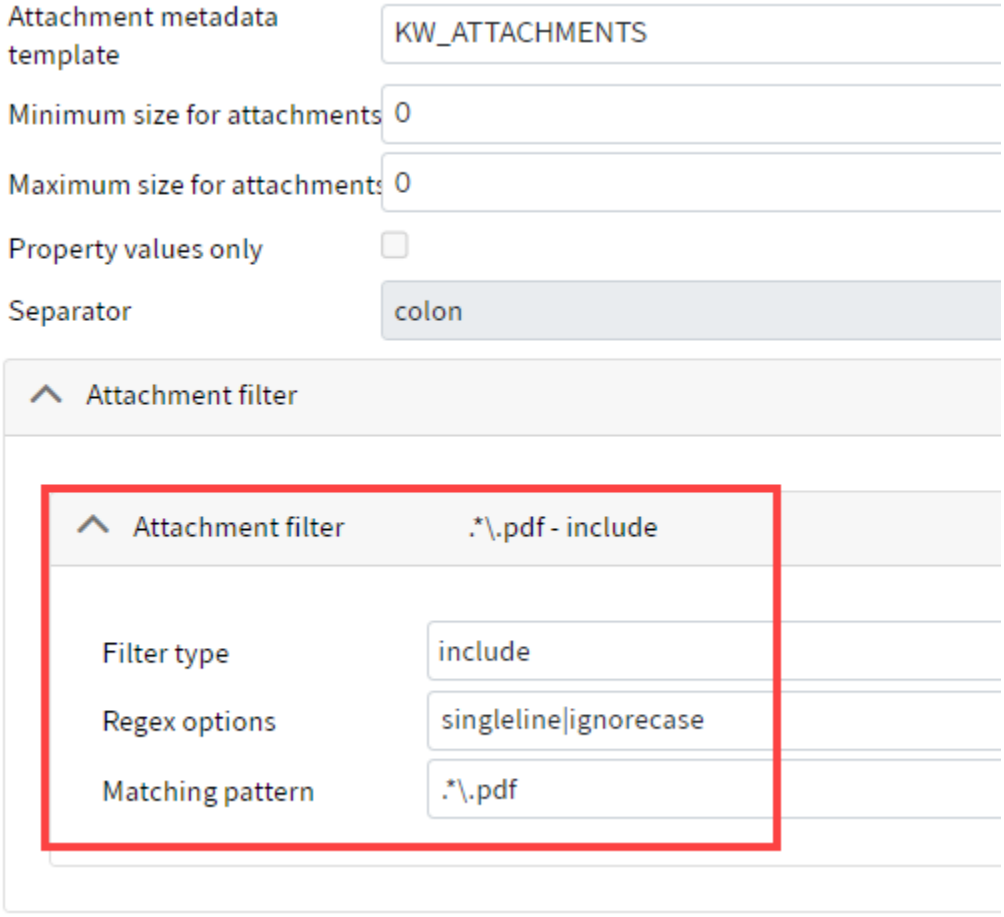
Test this using the last message as an example. You could also vary the example by changing the attachment path and selecting *Go to* in the task view after filing with the respective workflow and then checking which folder you are taken to.

For the last example, you need a message with two PDF attachments and any other file attachment. You can also embed an additional image as the fourth attachment and experiment with different export modes and filters once the example is complete. We recommend that you try out different parameters in a test environment to help you learn and understand the processing capabilities of ELO XC.

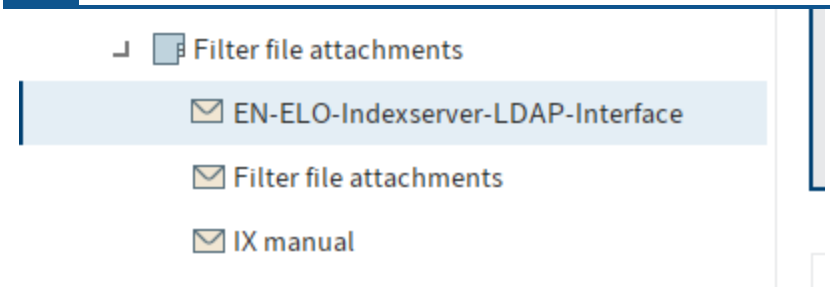
The test message now looks like this:



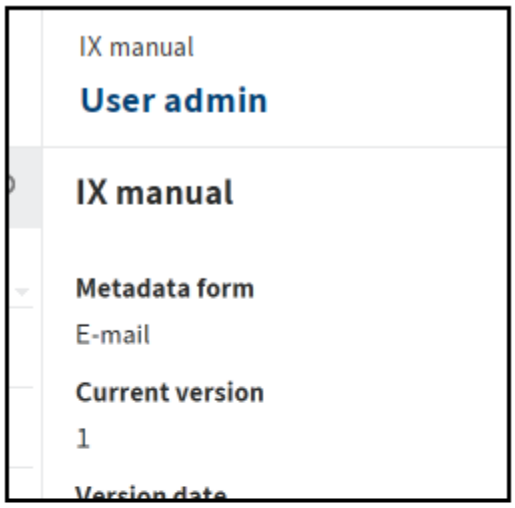
To extract the PDF attachments, store them, and forward them to a workflow, it is a good idea to filter by file extensions in the *Export* action.

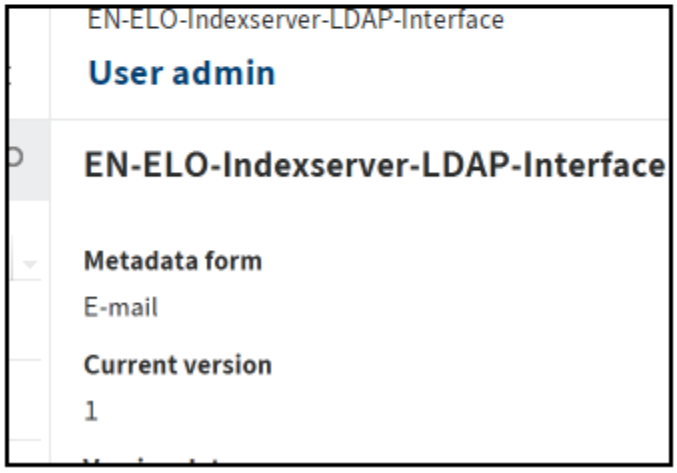


You can use the *Attachment filter* list to check the names and accordingly decide whether to *include* or *exclude* the attachments. Regular expressions are used to check the pattern. Change your export settings as required. Before you start processing, make sure that the *EloAttachmentGuids* property is set in the *External call* action. After processing, you should see this result in the repository:

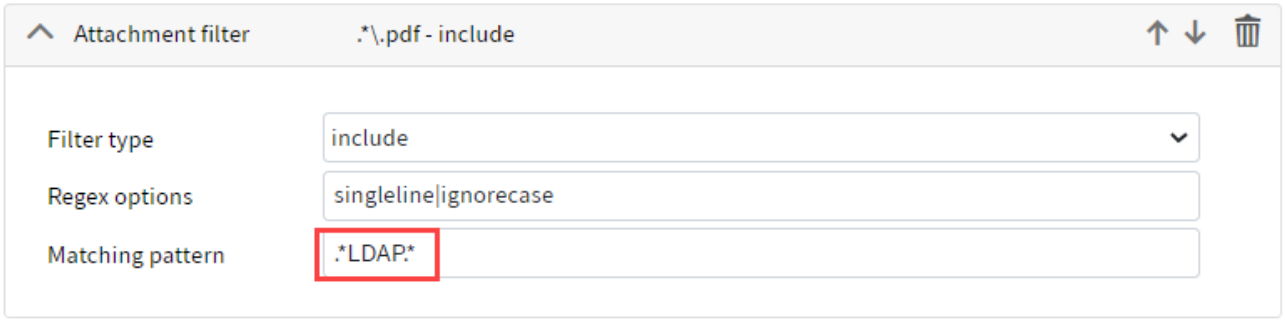


The two associated workflows also show up in your tasks:

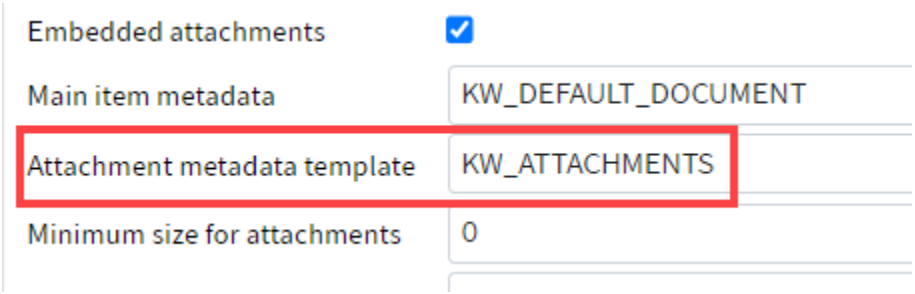




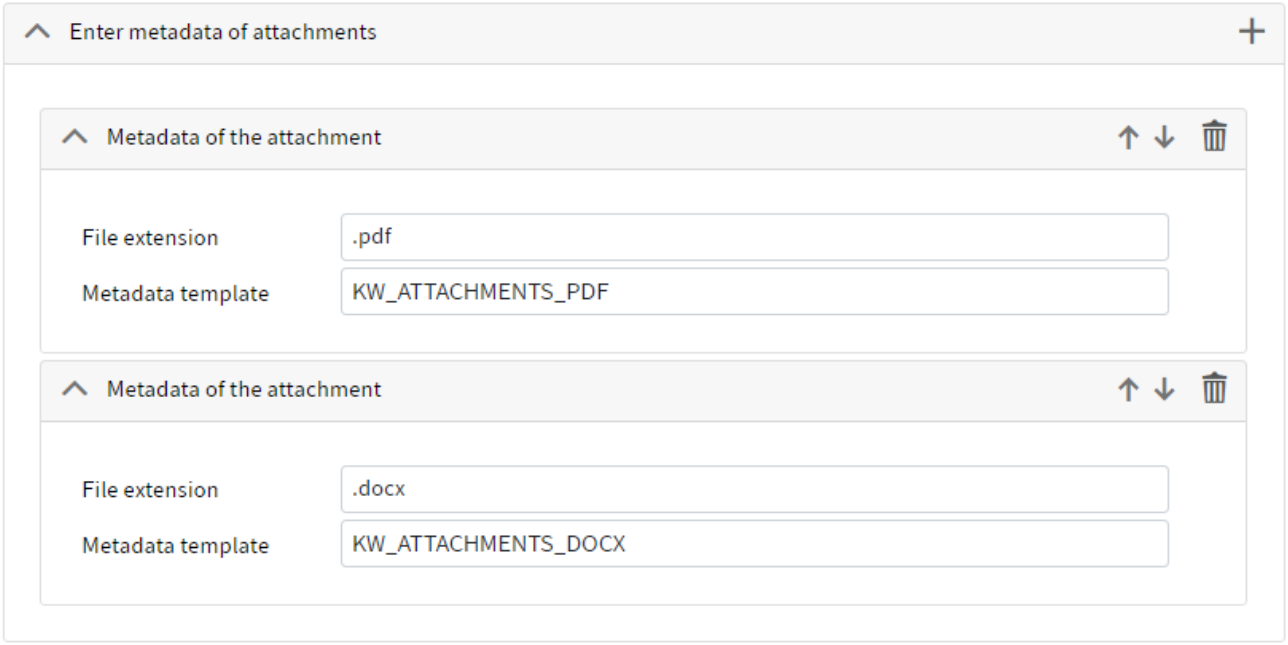
However, if you want to store the two attachments that have LDAP-related content, you need to change the matching pattern in the *Export* action:



After processing, you will find both the PDF and the DOCX file in the repository, and you will notice that both documents are using the same metadata form. It is configured in the *Export* action using the appropriate parameter for all file attachments (*KW_ATTACHMENTS* was chosen here):



However, it is not always the best approach to assign different file attachments to a single metadata form. The PDF could be an invoice, and the DOCX file could be a technical specifications manual. The invoice needs to be routed to accounts, and the other file is treated as documentation. These two different purposes are likely to require different metadata forms. You can change the configuration in the *Export* action to ensure that different file extensions are assigned to different metadata forms when they are stored:



Information

The file extension used here is merely the extension of a file name and is intentionally not a regex pattern.

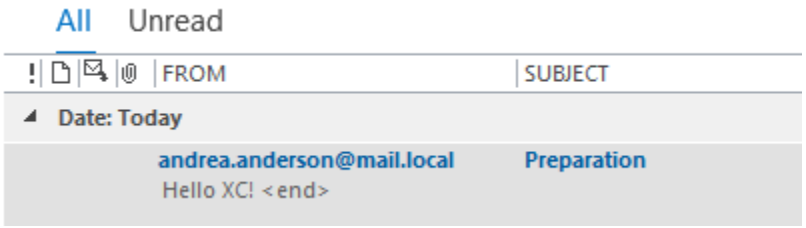
Link message parts

In this section, we show you examples of how to link separated message parts in the repository when storing messages with ELO XC. You can then use these scenarios as a basis when you have a similar scenario in a project.

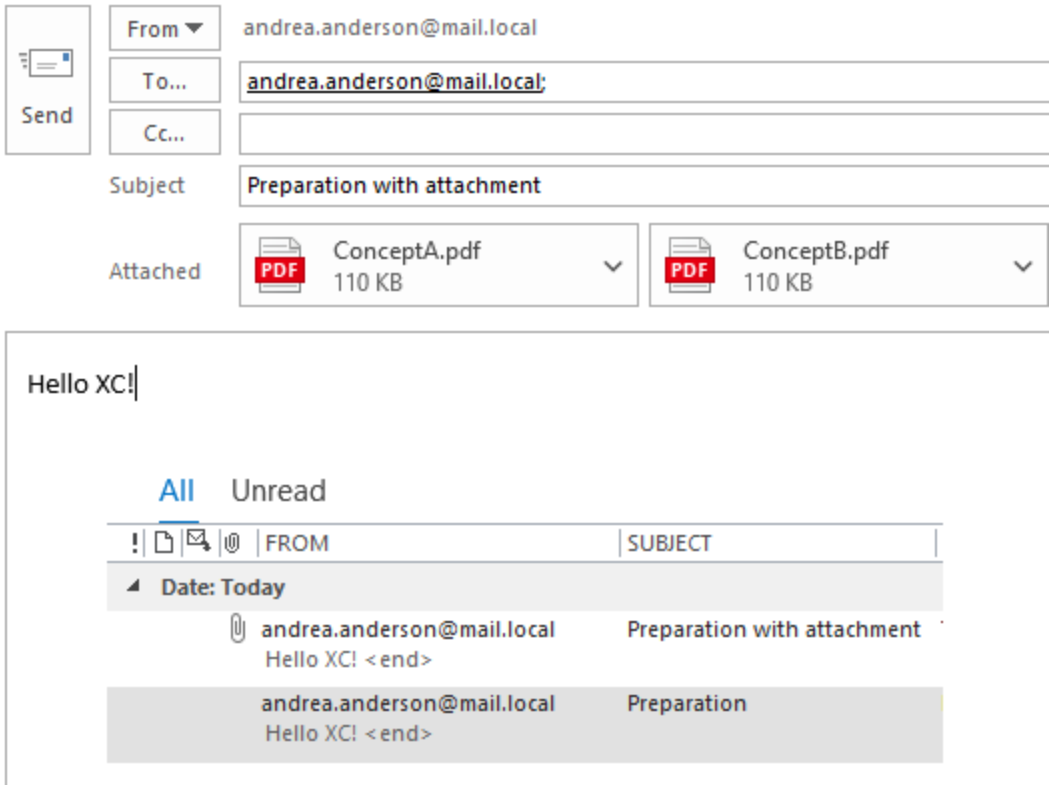
All examples are based on a core feature of ELO XC, which enables you to split e-mails and store their attachments separately. You will find this setting in the *Export* action. Besides the physical separation aspect, you also need separate filing paths, which you configure with the *Filing path* action. The configuration of links in the repository is done in the *Archive* action.

Preparations

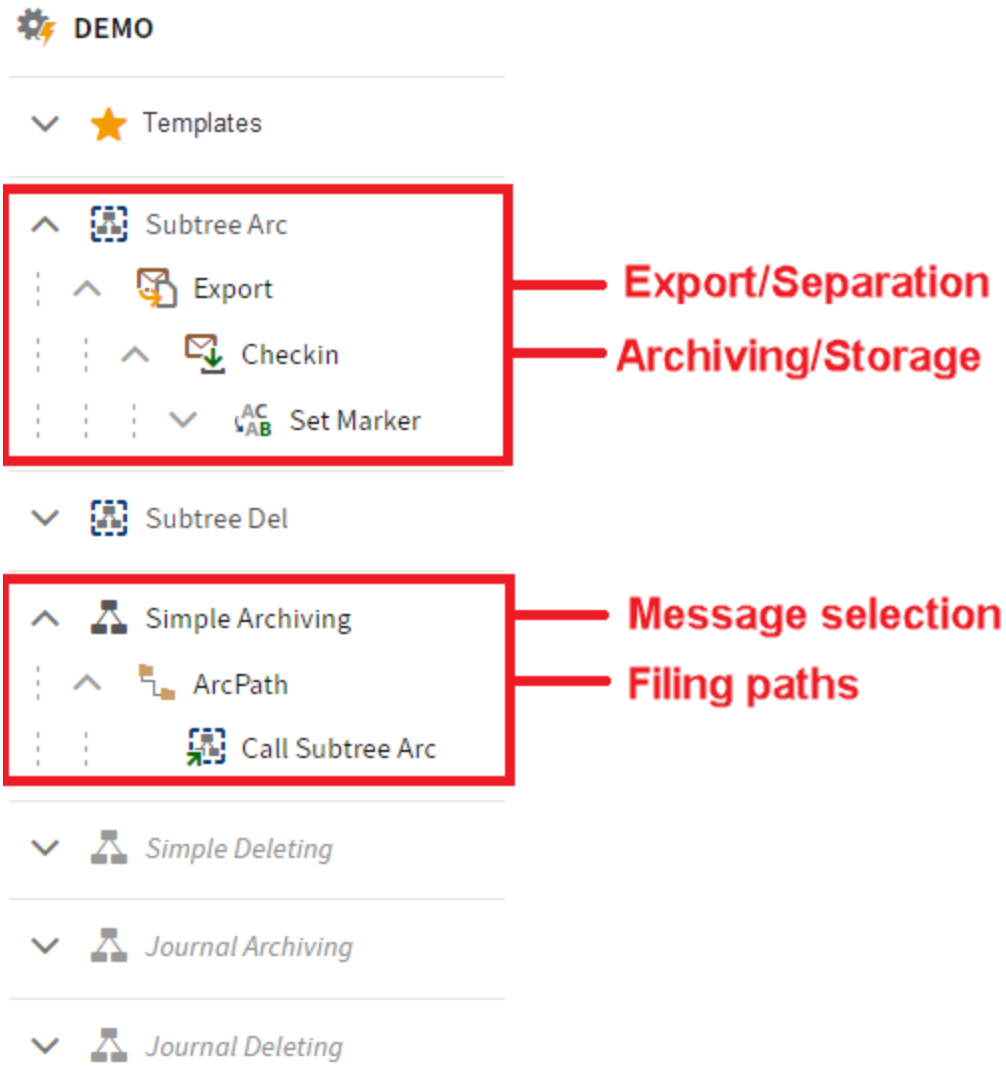
We have used an Outlook connection to a local mailbox for all examples. This is sufficient to illustrate the use cases. The mailbox sends all messages to itself because we do not require different recipients in the examples:



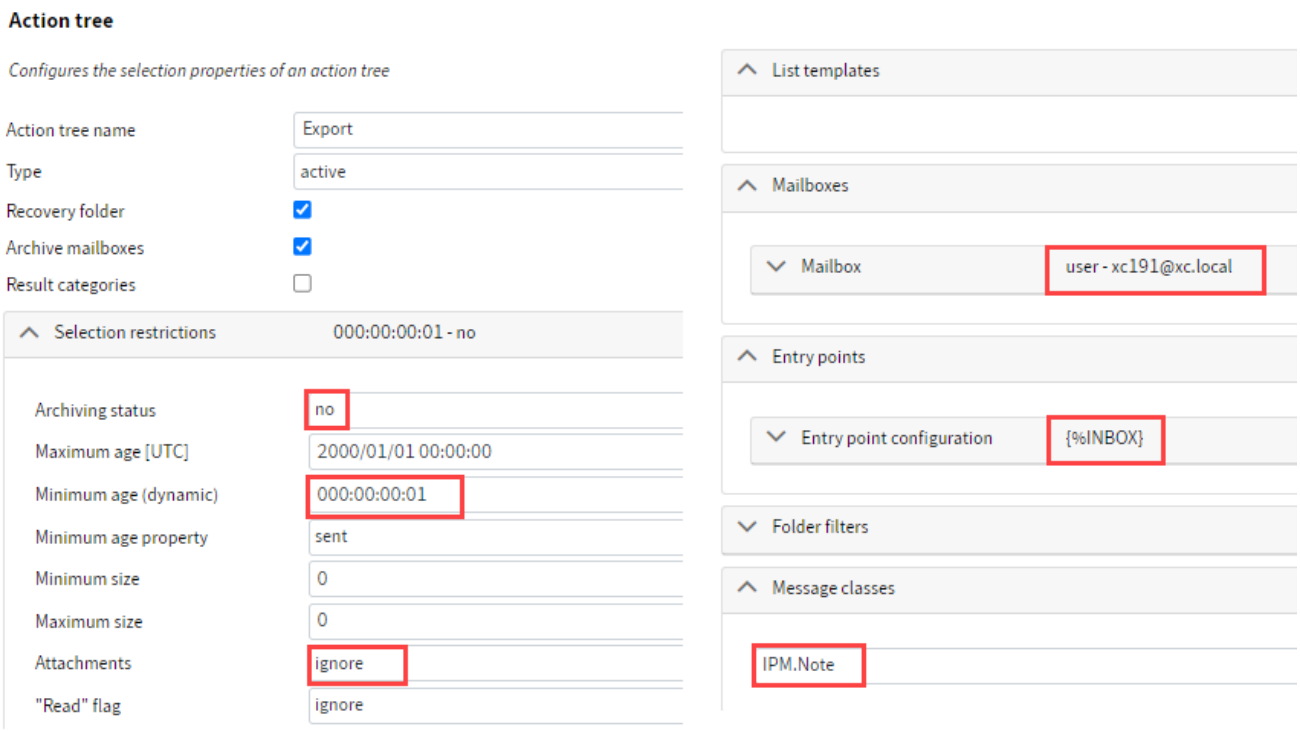
To create the e-mail, we need to prepare two files that we will repeatedly use as attachments:



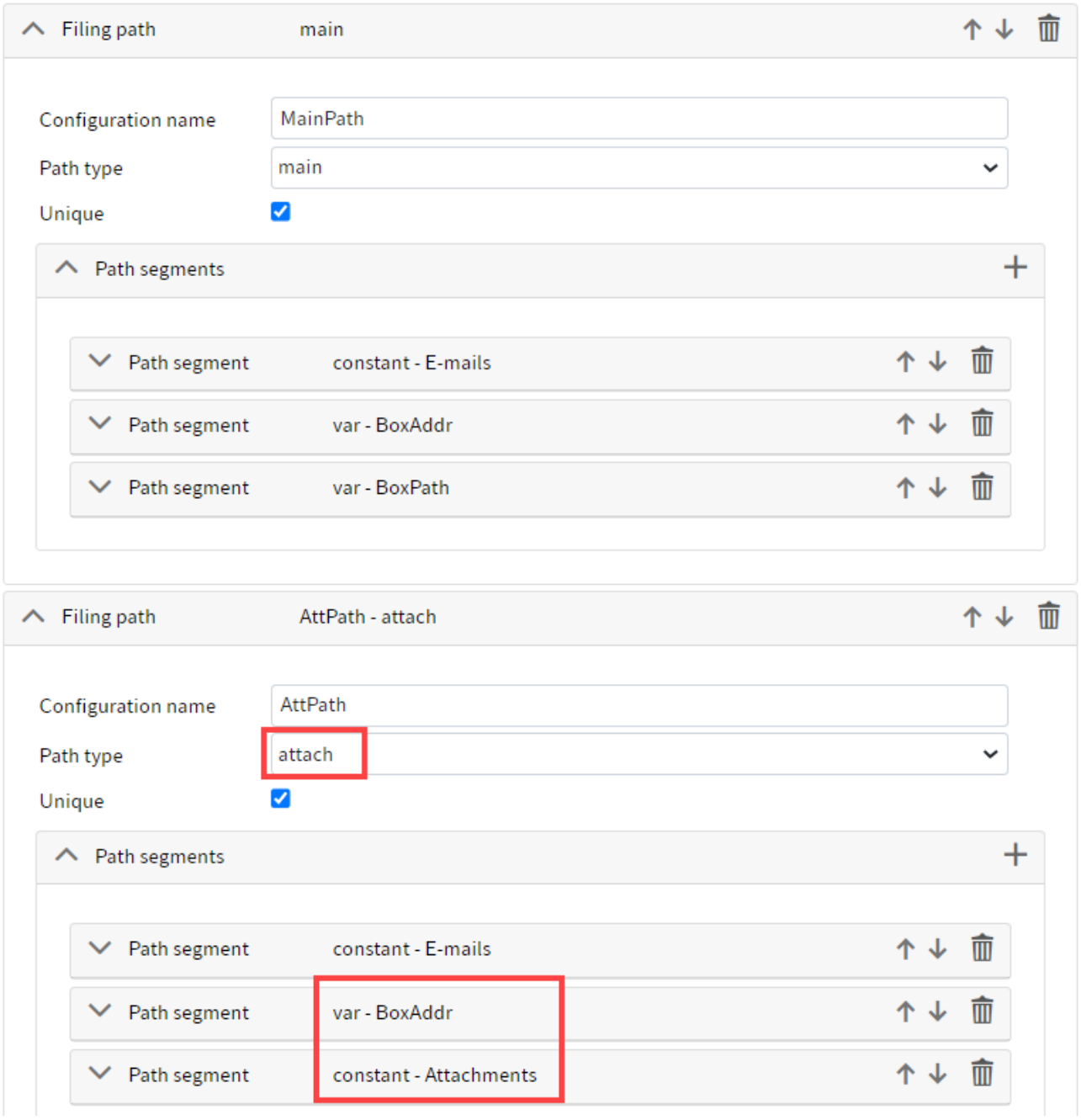
An automatically generated instance is used for the basic configuration of ELO XC. The focus is the action tree for selecting messages as well as the associated subtree that contains the *Export* and *Archive* actions:



Set the execution mode of the instance to *idle*. This means that the instance will only process jobs if started manually. The *Simple Archiving* acrtion tree is the only active tree in the instance. It selects all e-mails in the inbox for processing if they have not been archived yet:

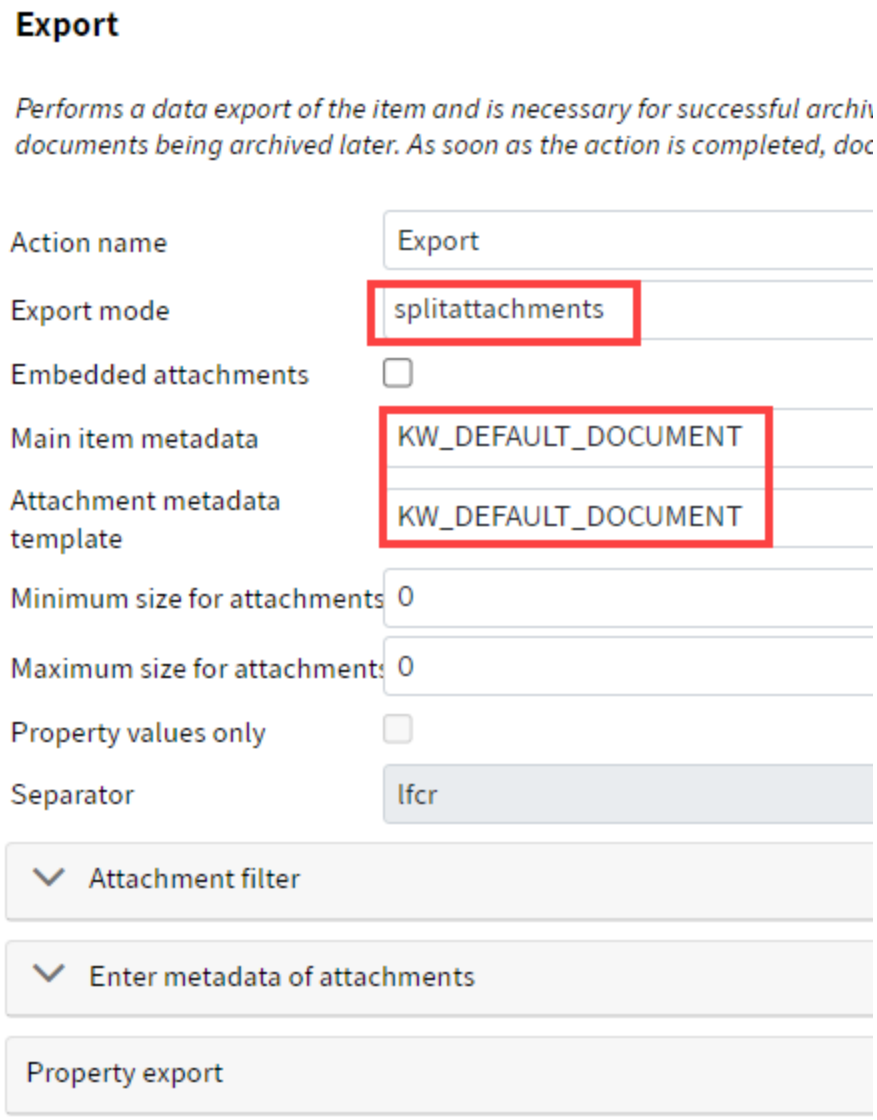


The selection criteria do not explicitly include attachments because we are looking for a configuration that works for messages with attachments as well as for messages without attachments. To ensure that the extracted attachments are also stored separately in the repository, we need to configure a second filing path:



While e-mails should be stored in the folder of the mailbox in the mailbox path illustrated, all attachments of this mailbox should be archived in a collection folder.

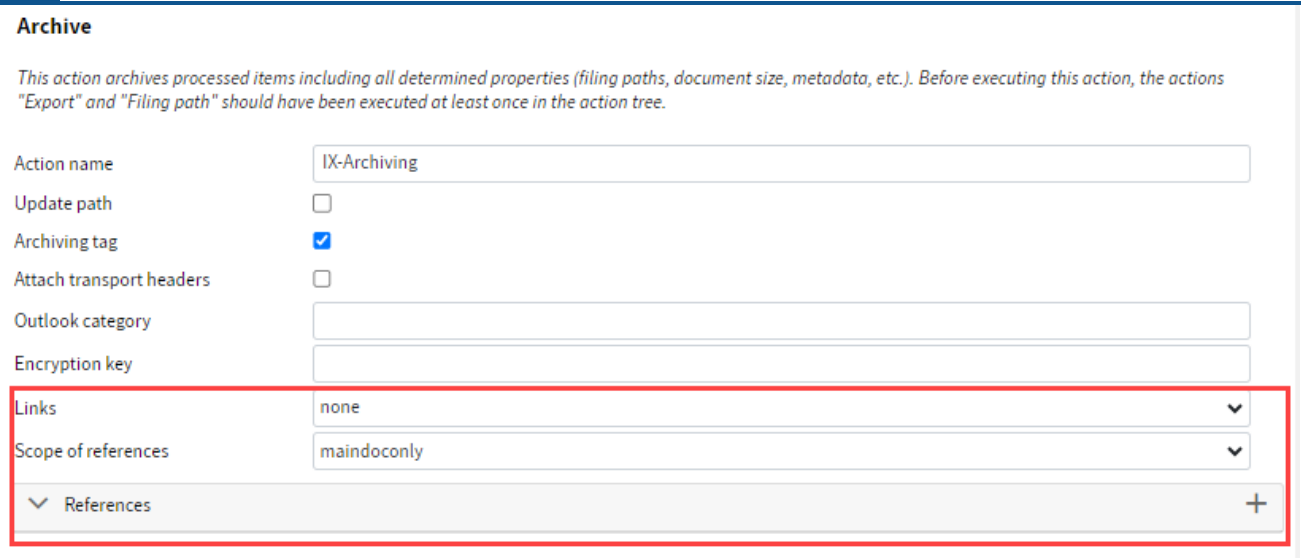
The export action separates the attachments:



To extract attachments, we could have chosen *split* or *attachmentsonly* in addition to *splitattachments*. In our scenario, we assume that the messages will be stored in their original state, since we cannot predict at this point in time in what way we will need them in five, ten, or 20 years. This is why the original state takes precedence over forced optimizations of the physical memory. However, we still want to store the attachments separately so that we can use them in the repository.

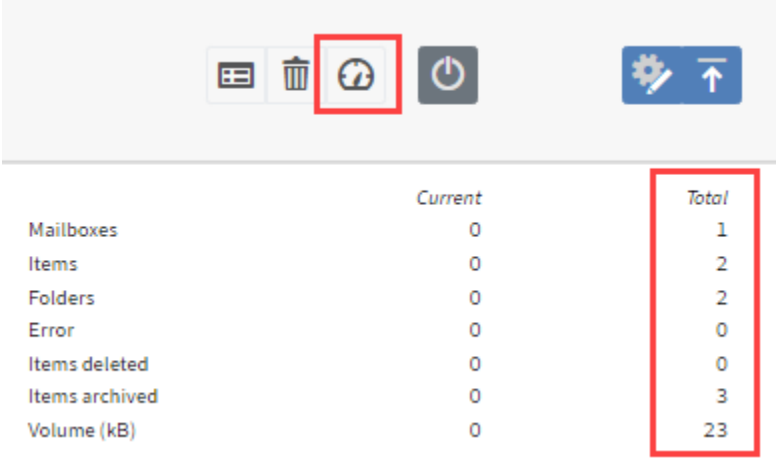
The default metadata template is also useful for attachments. This allows us to see in the repository that a document was originally a attachment, and we can still see the original recipients. Data context is especially important in the long term, when the origin of documents is no longer evident. Therefore, we do not need metadata templates depending on the attachment or additional filters for attachments.

The *Archive* action, which stores documents in the repository, is the main action used in the examples in this section:



The links to the stored documents are configured using the options highlighted here. We will introduce these parameters in the following examples. Now you can take another look at your instance configuration to check if you have done the right preparation.

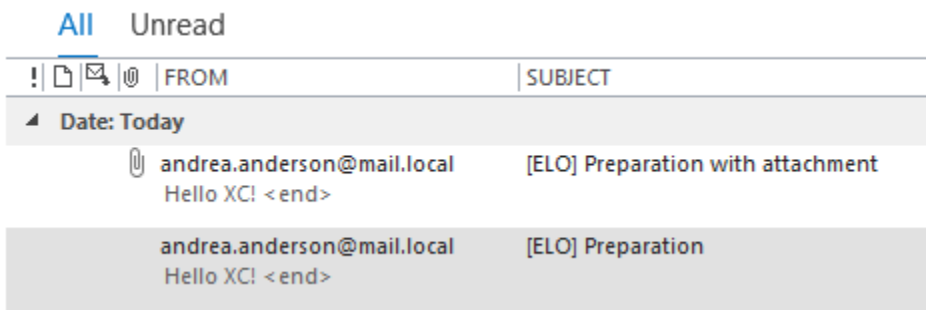
If all settings are correct, you can process a job in ELO XC. Open the processing statistics:



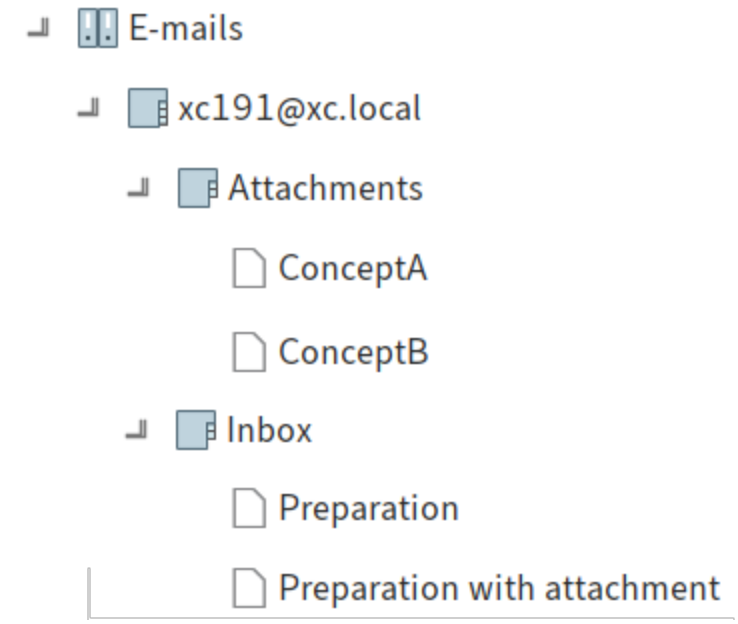
	Current	Total
Mailboxes	0	1
Items	0	2
Folders	0	2
Error	0	0
Items deleted	0	0
Items archived	0	3
Volume (kB)	0	23

You should see that exactly two items were processed in a mailbox folder. Four itens were archived – two e-mails and two attachments. You can check this in the following examples at any time.

The e-mail subject in the mailbox has a tag assigned to it:

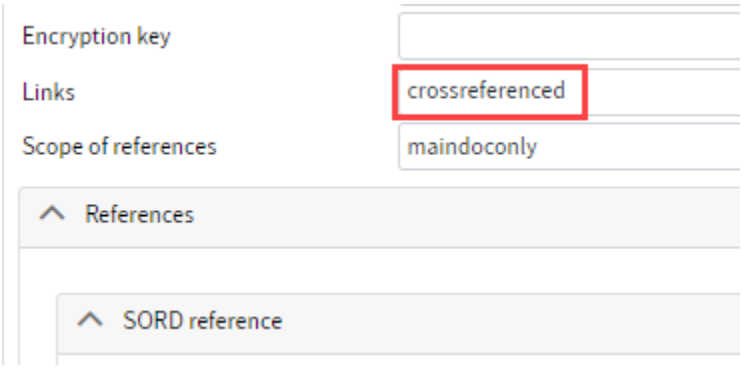


You see the configured filing structure in the repository:



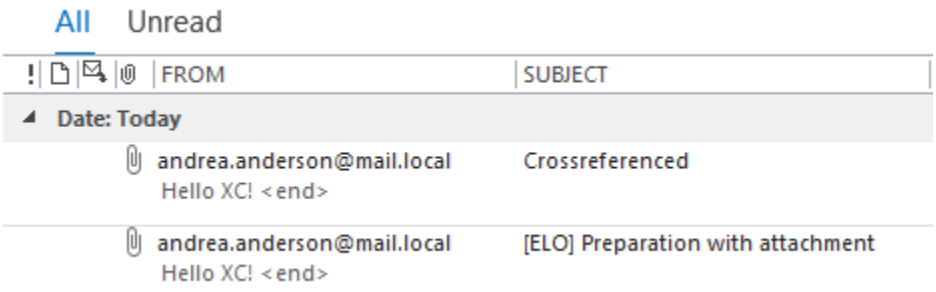
SORD links

SORD links are required to associate documents with each other:

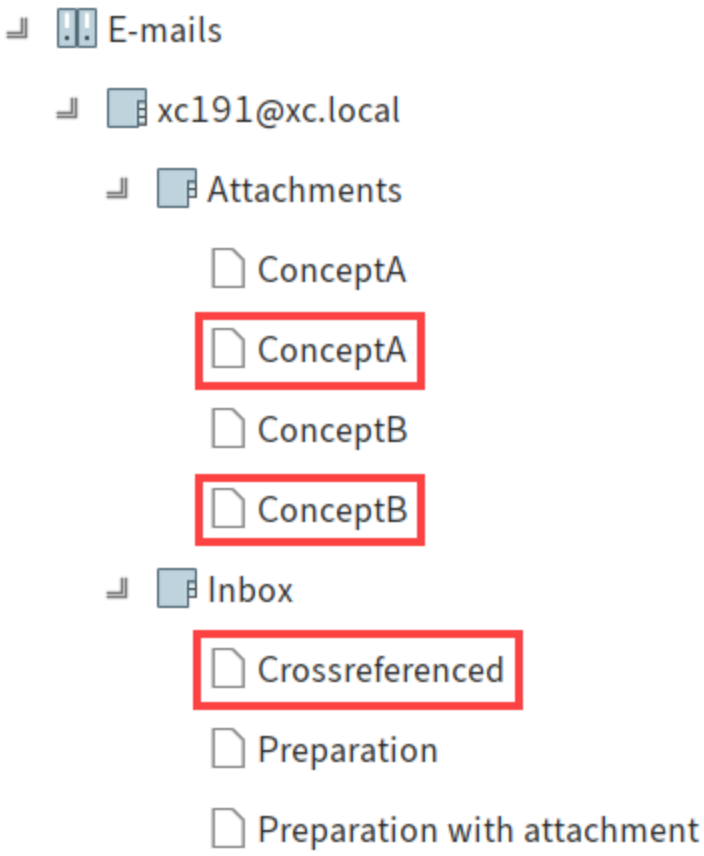


You can choose between *crossreferenced* and *permanent*. The difference is that permanent links cannot be removed: *crossreferenced* is called a *mesh link*, *permanent* stands for an inseparable, *permanent link*. ELO XC selects the technical names from the Indexserver interface.

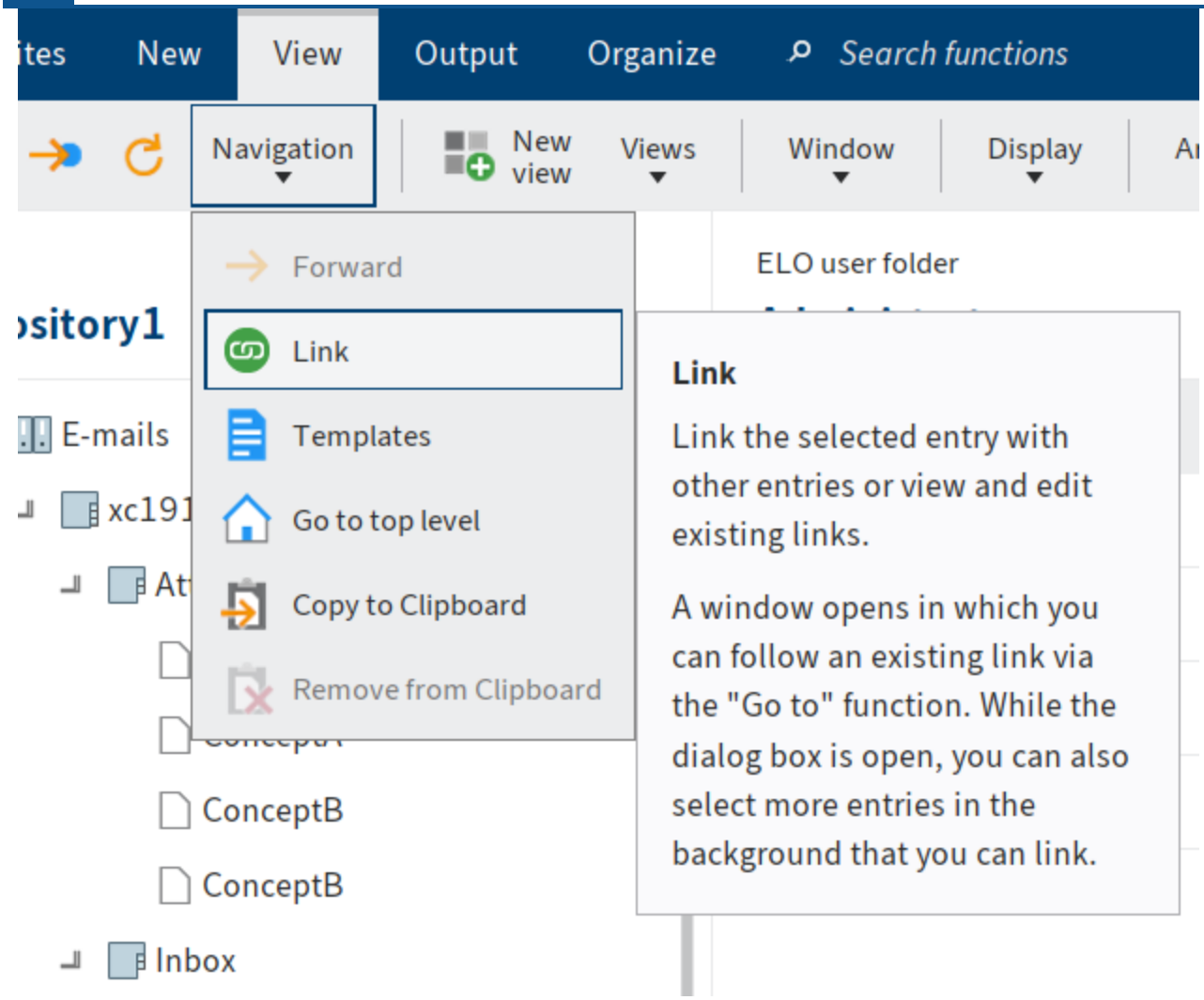
Create a new e-mail with `Crossreferenced` as the subject and add both attachments:



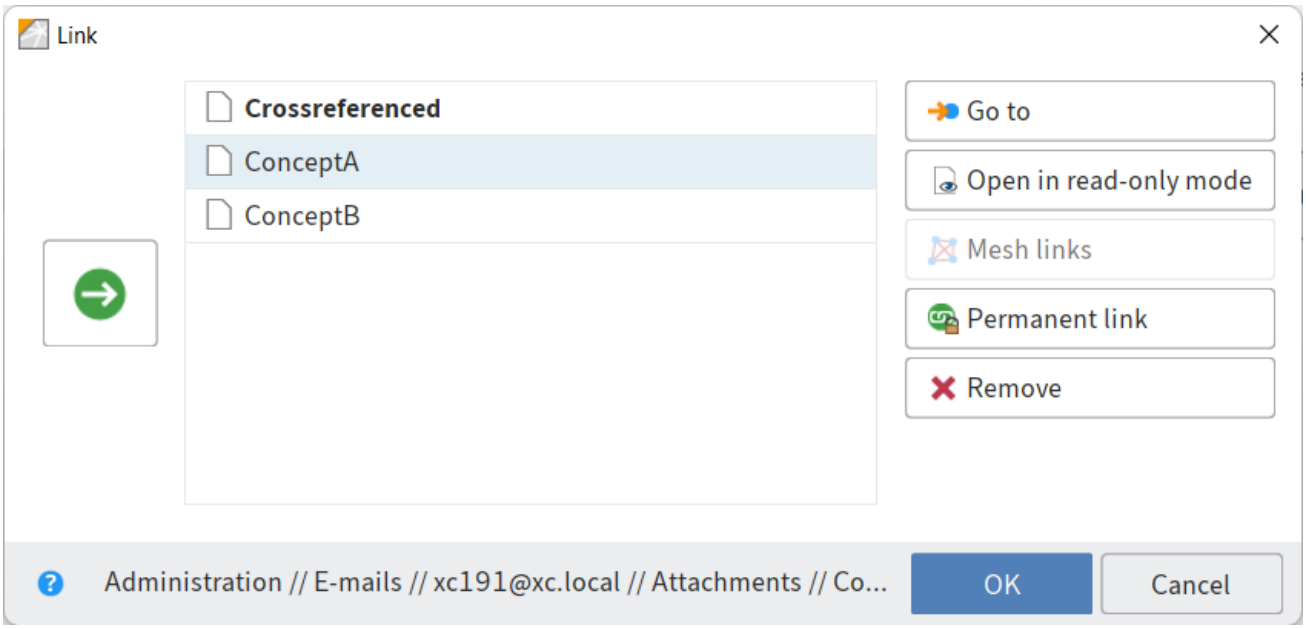
Process the *Crossreferenced* e-mail with ELO XC. You will see the following result in the repository:



As expected, three documents were stored and associated with links. To check this, you can display the link in the client:



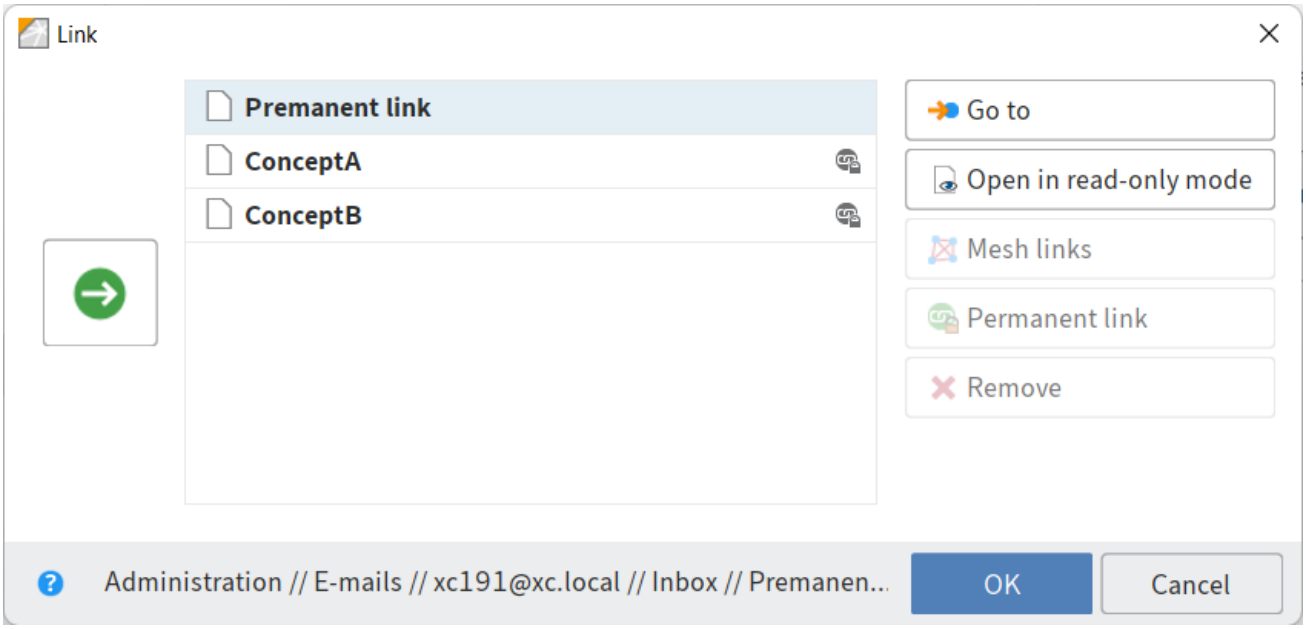
Select *Crossreferenced* and open the *Link* window:



When you select one of the entries, you will notice that you can no longer mesh the links. This is because of the settings made by ELO XC while the links were being generated. However, you can still delete the links or convert them into inseparable links.

Create a new e-mail with the two attachments and call it `Permanent link`. Next, set the SORD links in the *Archive* action to *permanent* and process the *Permanent link* e-mail with ELO XC.

You will find the stored message parts in the repository, but after opening the links you will see the difference:



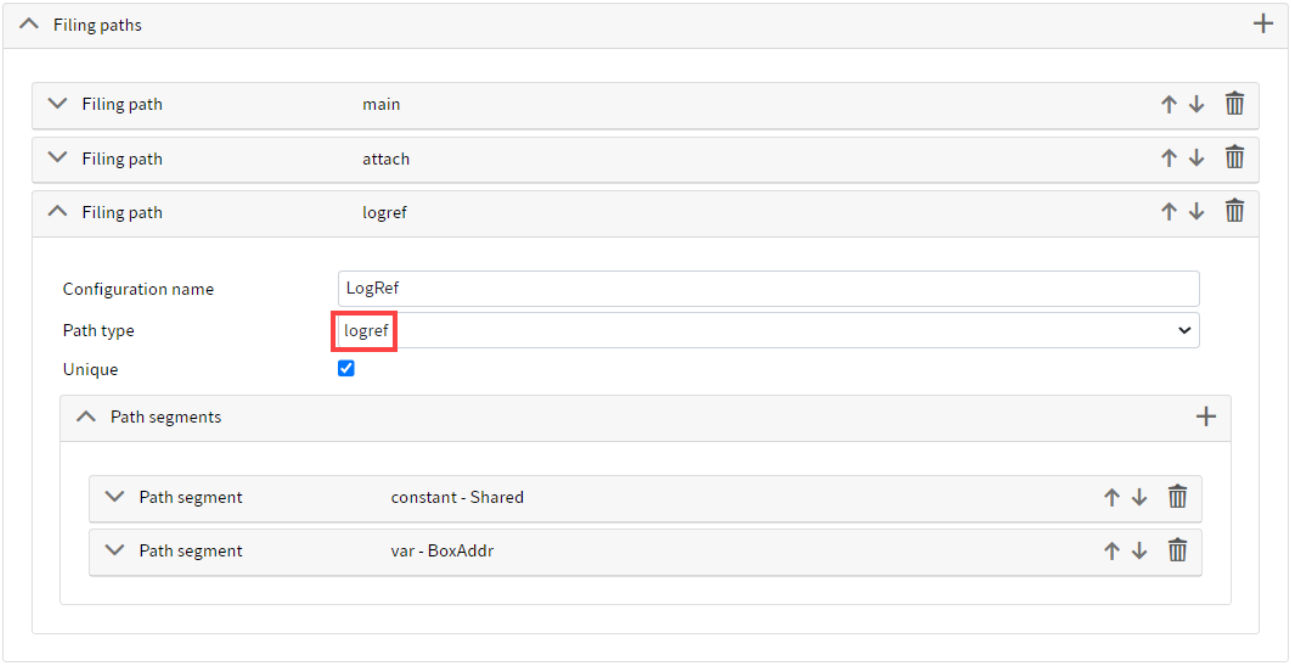
These links cannot be deleted and are already permanent mesh links. This means that the *Permanent link* message parts are irrevocably associated with each other.

> **Information**
>
> Assume you have a large repository containing several million documents that is accessed by more than 500 users on a daily basis. It is likely that over the years, the documents will not remain tied to one location in the repository. In particular, it is probable that extracted attachments will be moved around in the repository. In addition, if filing or folder structures are reorganized at some point, it is possible that users won't be able to find the attachments that were once separated from the original e-mails. By permanently associating message parts with each other using SORD links, you no longer need to take steps to relocate message parts that can no longer be found. Especially in the case of permanent, inseparable links, you can also identify the affiliation of message parts in the long term.

Finally, set the SORD links value back to *none*. Delete all documents stored in this example from the repository before continuing with the logical reference example.

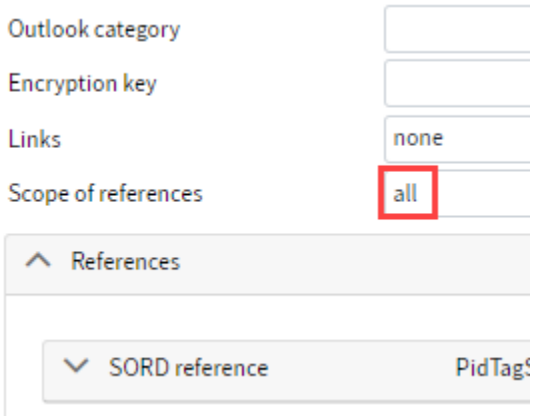
References by filing path

We are going to use the instance configuration that we created in the Preparations section for this example. The e-mails will be stored in the repository using the paths configured for this purpose. We will show you how to easily create logical references for stored message parts. First, we create a new path entry for logical references in the *Filing path* action and configure the *Archive* action for the logical references type:



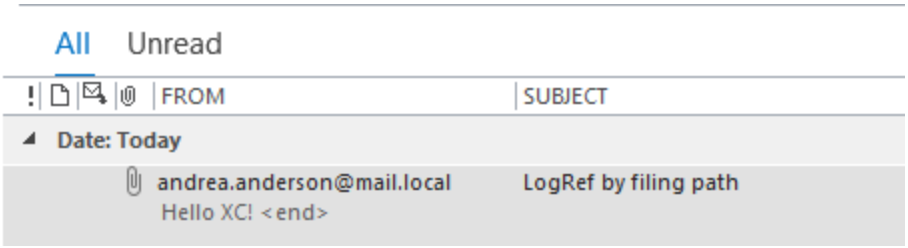
We now add a *logref* type to our filing path. As a result, the *Archive* action will automatically store all created logical references in this path. In our example, we assume that the mailbox folders with the originals in the repository can only be accessed by the respective mailbox owner. However, other users should be able to read the e-mails. To allow this, we create a folder with logical references to the originals in a *Shared* folder for each mailbox. The *Shared* folder with read rights for all users has already been created in the repository.

In the *Archive* action, we configure the type of references:

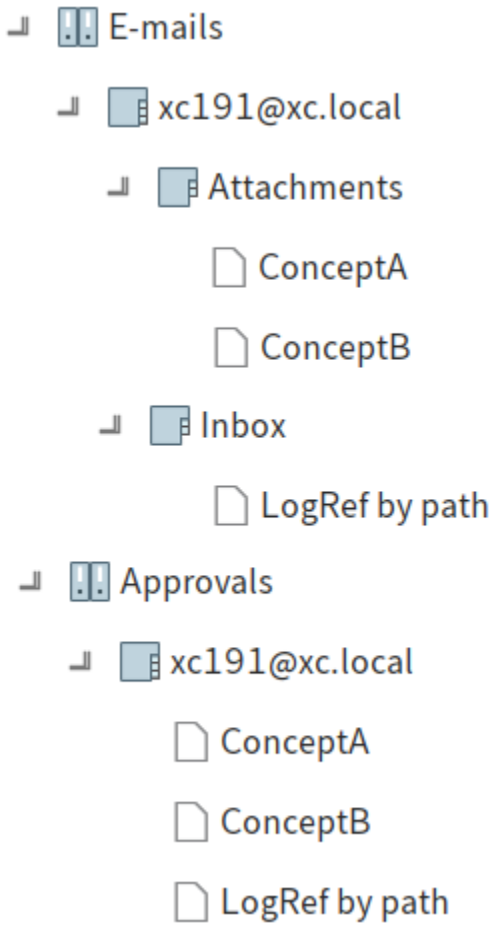


The *maindoconly* type only creates logical references for the e-mails themselves. The same applies for *attachmentsonly*. With the *all* type, all message parts are referenced.

Modify, validate and publish your configuration as shown here. Then create this e-mail with an attachment:

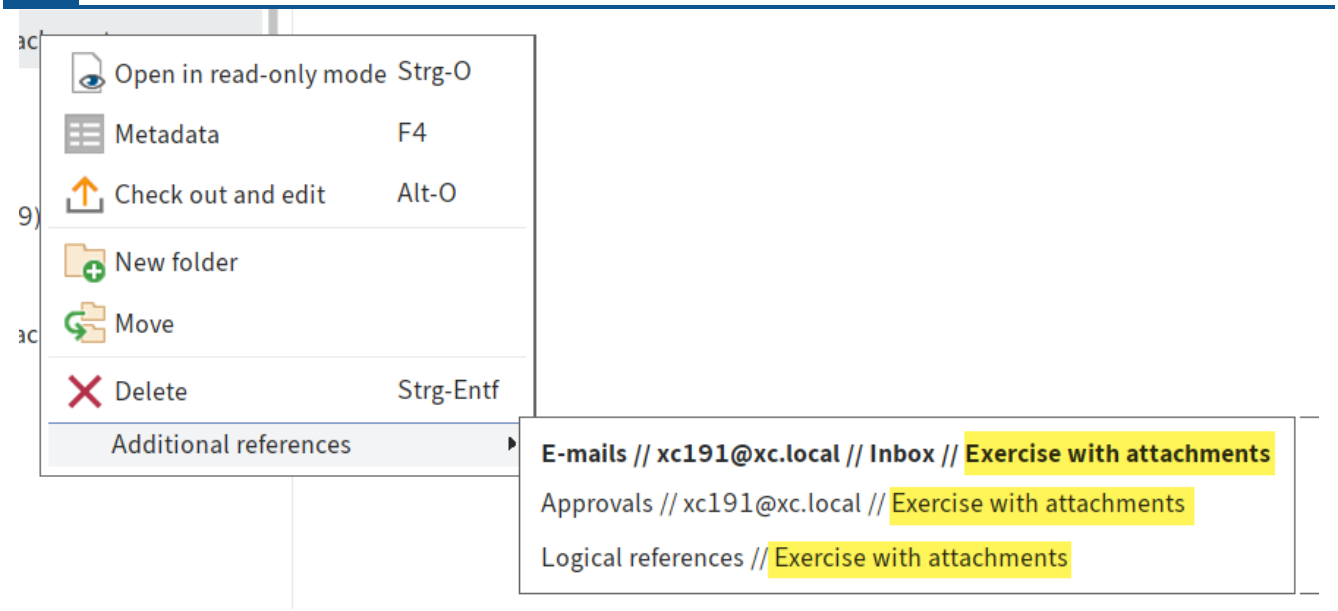


When you start processing, the message parts will be stored as documents and additional references to the documents will be created in the *Shared* folder:



We'll end this simple example with an exercise:

Create two new e-mails called *Exercise* and *Exercise with attachments*. Modify the configuration so that you obtain the following result in the repository (example for second e-mail):



You can still invent your own paths for logical references, test them, and create references for just attachments.

Before you continue with the next example, delete the paths for logical references from the *Filing path* action. Clean up the repository and your mailbox as well.

References by metadata field

We are going to use the instance configuration that we created in the Preparations section for this example. We'll illustrate the process of creating logical references using metadata with an imaginary scenario that could actually occur.

For this scenario, we are going to create a filing structure for a specific company project. The company builds dams and wind turbines. E-mails are stored separately for each mailbox but the client wants ELO XC to additionally generate a project association and make the e-mails visible and/or usable in the project folders. The next section contains some preliminary considerations in this context.

Preliminary considerations

We could configure an e-mail filing system that recognizes that an e-mail belongs to a project while processing it. Based on this, we could modify the filing path. However, this means that the original messages are not stored in the repository folder of the associated mailbox, but in the project folder in the repository. Of course, there are scenarios where this approach may make sense, especially if it is a project mailbox that is not *associated* with a specific project stakeholder.

If this association with a specific user exists and the mailbox owner needs to be able to access the e-mails in their own repository folders, we could store the project e-mails twice by configuring the action tree to create a second after the e-mail is initially stored and then storing it again in the new path. The company would need to assess whether the project is important enough to justify the

additional memory this method requires. The use of physical MD5 paths (*passive duplicate control*) could effectively reduce the amount of physical memory required.

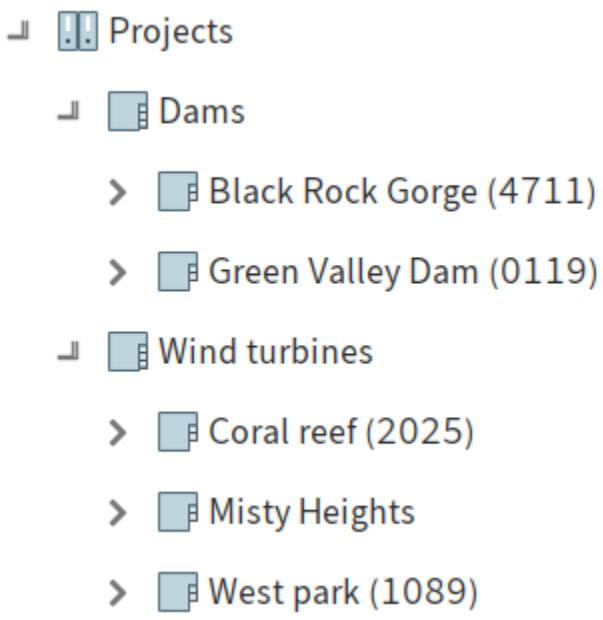
However, we are also going to assume that mailbox owners are going to change data after the e-mails and attachments have been stored. If we were to create duplicates, we wouldn't know which were the newest changes and how they could be automatically reproduced in the projects.

Example: The mailbox owners assigns additional metadata to project e-mail after storage. This change is relevant for all project stakeholders. Does the mailbox owner have to assign this metadata twice? Perhaps this can be justified in exceptions, but if lots of different mailbox owners do this for different projects, it is bound to cause organizational problems due to different versions of duplicates being created.

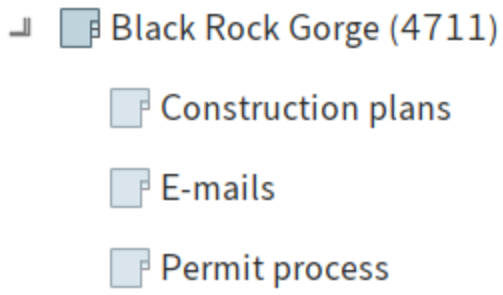
Logical references are a good compromise. Storage of the message parts remains prevalent as it can be assumed that not all filed messages are relevant to a project. However, if an e-mail is relevant to a project, it is possible to refer to its attachments in the project folders. As subsequent changes are only made to the original documents, these are automatically applied to other versions. Moreover, this also means that the changes don't have to be made twice.

Project folder structure

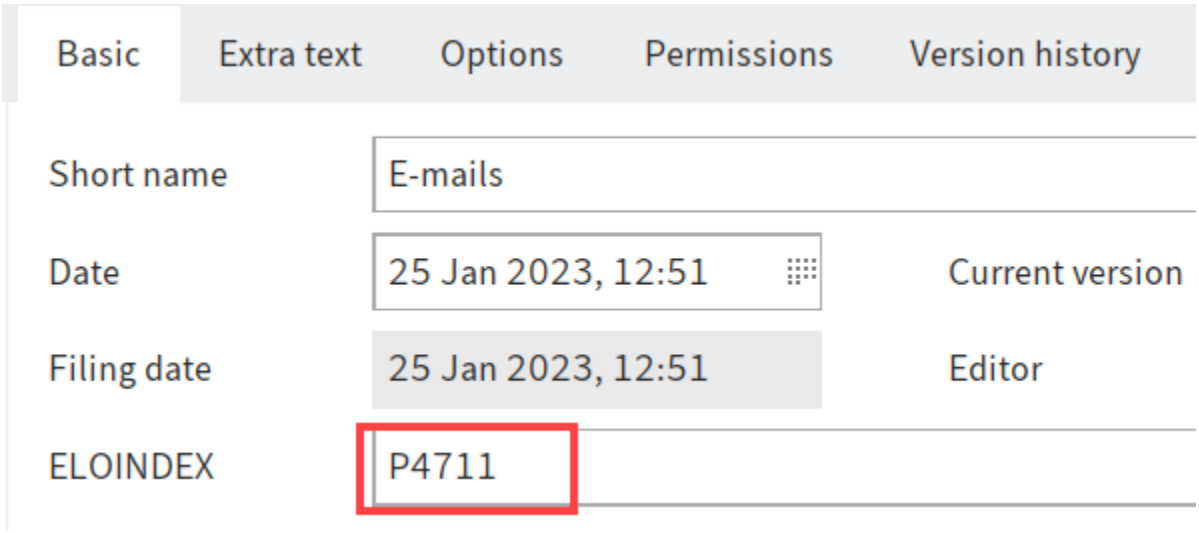
Before we configure the logical reference example in ELO XC, we need to create the project folder structures in the repository:



The projects are split up into dams and wind turbines. Each project has a project number in brackets for unique identification (e.g., correspondence, accounts payable, general administration). The project folders also have a more or less standardized child structure:

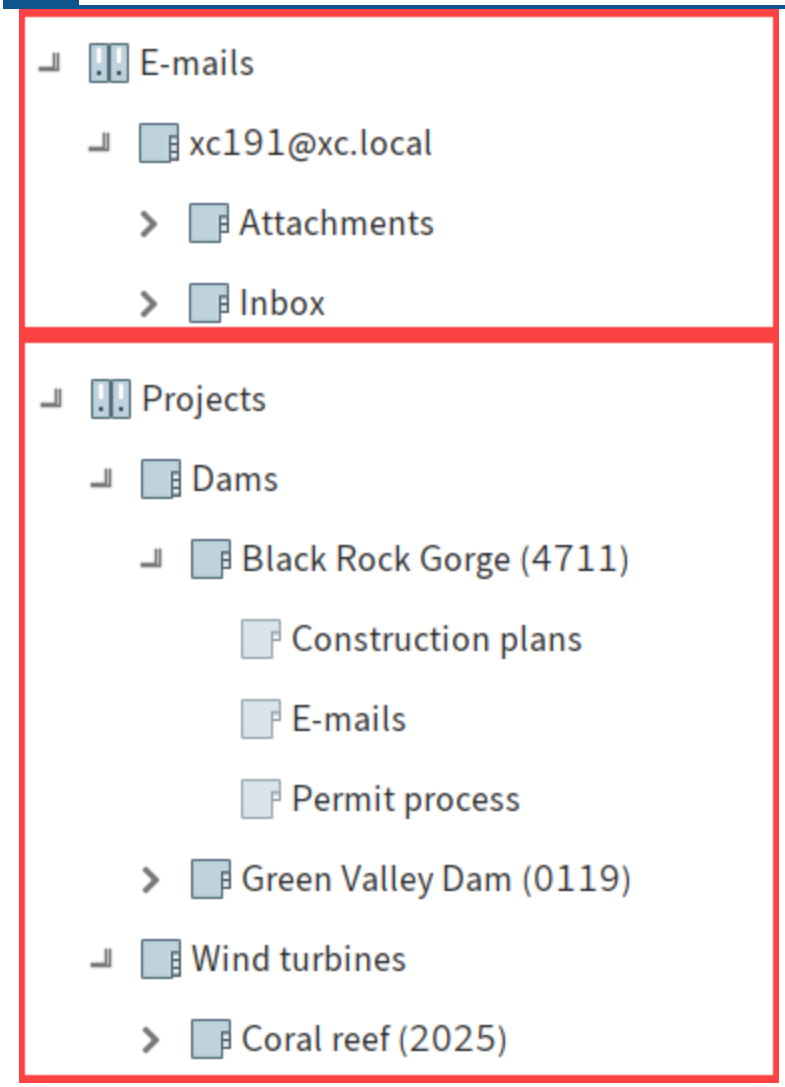


ELO XC does not necessarily require the names of all child folders to be identical. What is important is the project association, which is assigned with the help of metadata:



The project number is added to the folder for project e-mails as the *ELOINDEX* metadata field. The letter *P* stands for *project*.

Create the project child folders and assign each of the *E-mails* child folders the correct project number in *ELOINDEX*. The folder structure in the repository should end up like this:



You only see one open project folder in the figure, but all project folders should contain at least the *E-mails* child folder with the corresponding *ELOINDEX*. The documents are stored in the top part (*Repository of mailbox owner*). We create the logical references in the bottom part (*Project repository*).

> **Information**
>
> The correct SORD is determined using an index search via *ELOINDEX*. If multiple SORDs have the same value, i.e. the same project number, the SORD is chosen randomly from the set of possible results.

Project number

There must be a convention for project numbers to allow ELO XC to recognize project e-mails. In this example, the project number should be in the e-mail subject. It doesn't matter whether the number is shown in brackets at the end of the subject or as a prefix followed by a colon. It could also be in the middle of the subjext text.

The following variations are possible:

- `Progress of dam construction [PNR0119]`
- `[PNR2025]: Building materials delayed`
- `Conference [PNR0815] needs to be pushed back`

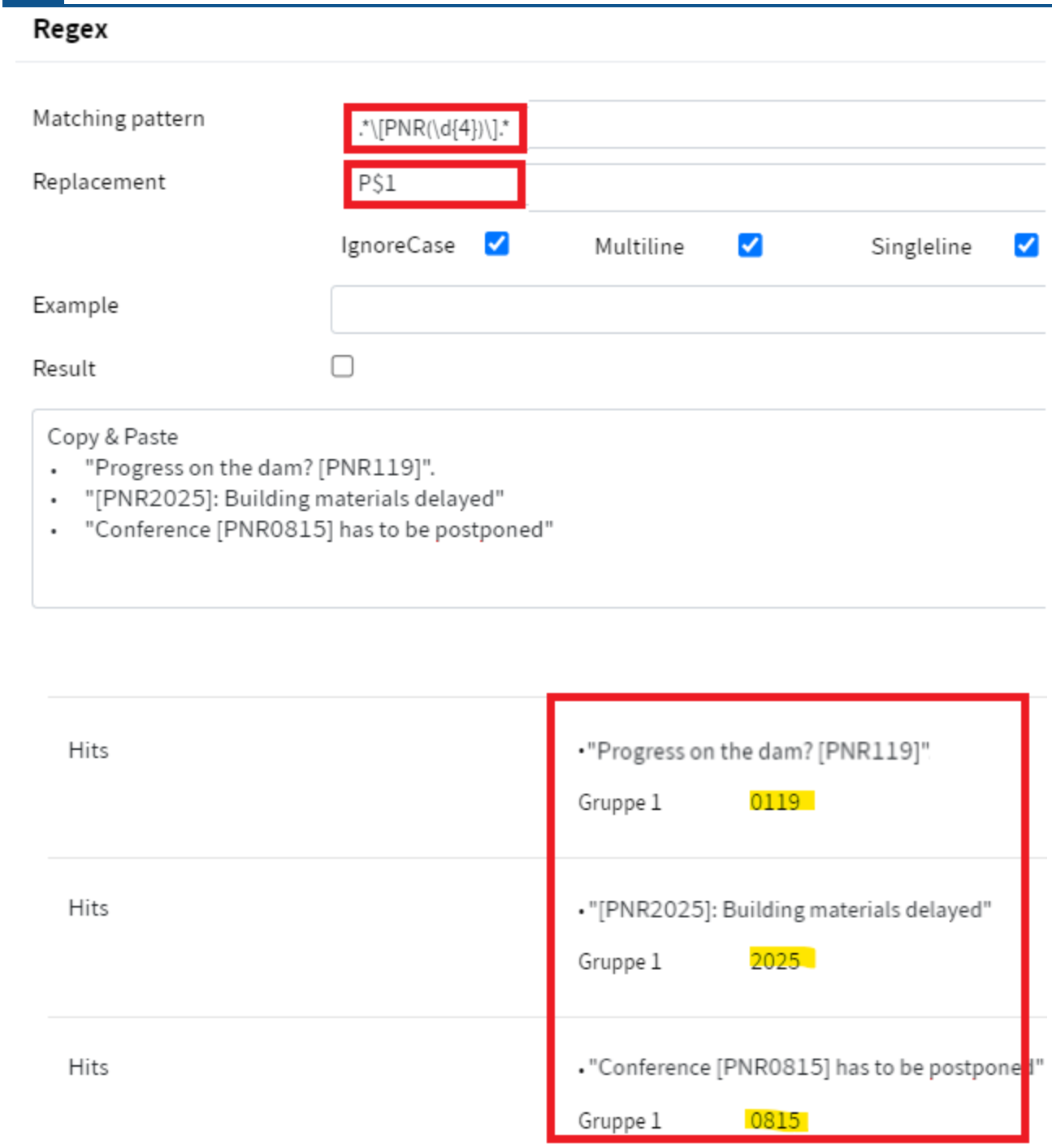
The rule for project numbers is as follows:

- The project number is always in the e-mail subject.
- It is enclosed in square brackets, has the prefix *PNR* for identification, and consists of exactly four digits.

To identify the project number, therefore, we need to evaluate the subject and find the project number, assuming it exists. This is done with the *Match/Replace* action, which can parse and transform strings using regex patterns.

However, we face the additional problem that project e-mails use the *PNR* identifier, while the project folder structure uses the letter *P* for the internal identifier in *ELOINDEX*. This means that not only do we need to recognize the *PNR* pattern, but we also need to transform it into the *P* pattern.

We solve the problem by first determining the regex pattern to configure the *Match/Replace* action. We need to use regex in the ELO XC tools menu:



You can copy the examples above to use in your own exercise. You should also use the *Multiline* option. The *Singleline* option will be sufficient for the e-mail subject later on. You don't have to worry about upper or lowercase.

If you enter the pattern `.\[PNR(\d{4})\].`, the four digits of a project number are bound to the first match group. Enter `P$1` as the replace value to generate the required project number format for the

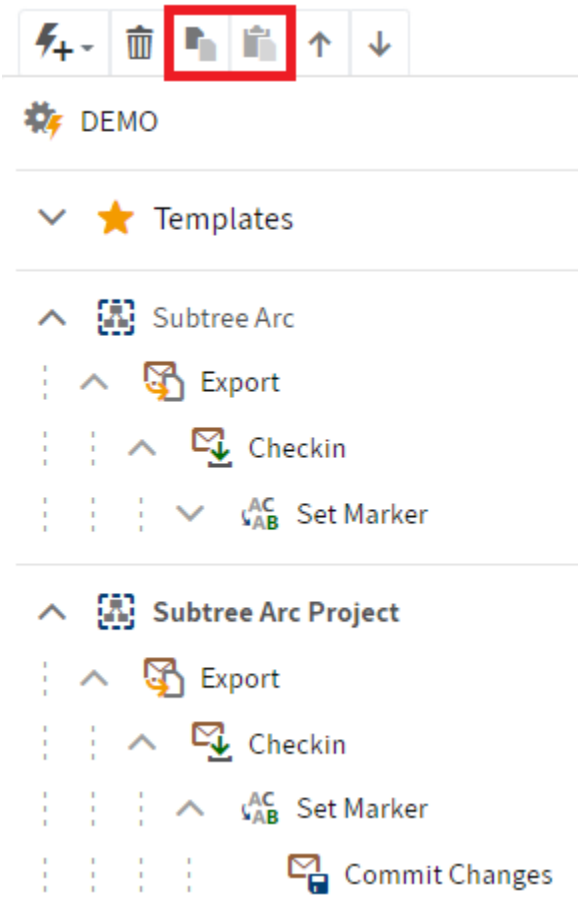
project folder structure. Click *Match* to display the match groups. When you click *Replace*, you should also get the following result:

Because the *Multiline* option was selected, the values were replaced in all input lines. If you can reproduce this result, make a note of the matching pattern and the replace pattern in a blank text document. We need both patterns later on.

Instance configuration

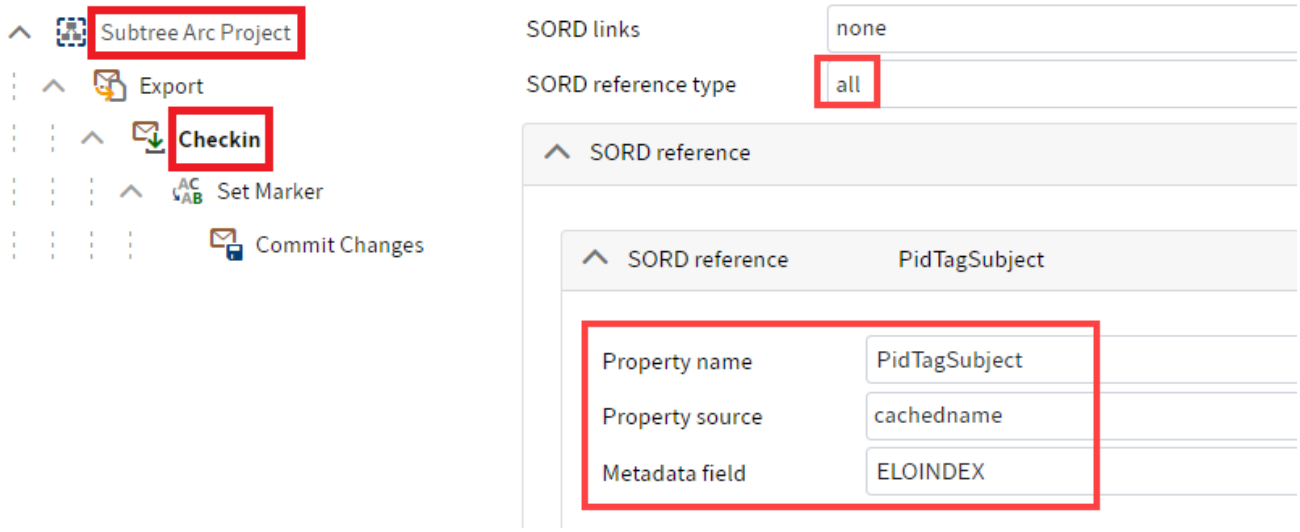
We now need to modify the ELO XC configuration so that the item is stored differently depending on whether a project number is found or not. If a number is not found, the item is stored as normal. If a number is found, the item is stored with logical references.

First, copy the subtree *Subtree Arc* and name the copy *Subtree Arc Project*:



Use the *Copy* and *Paste* functions to do this. Use the arrows to adjust the position of the new subtree.

Ensure that the *"Checkin"* archive action in the *Subtree Arc Project* subtree creates the logical references:



Set *all* as the SORD reference type. This means that references are created for all message parts.

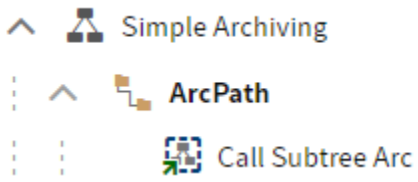
Now you need to set which property value must correspond to the metadata field value so that when it is stored, the logical references are created in the correct location. The e-mail subject is always *PidTagSubject*.

In this case, it is important that the property source is set to *cachedname*, since we will not change the original subject of the message under any circumstances and the converted project number is therefore obtained from the internal copy of the property.

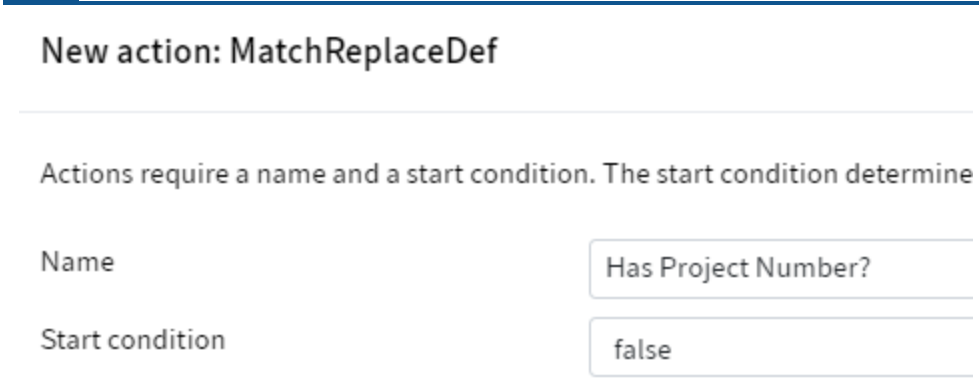
In other words, the generated project number of *PidTagSubject* must match a value in *ELOINDEX*.

Now that we have two subtrees, one with generation of logical references and one without, we have to make sure that the correct subtree is called. We therefore need to configure a case distinction before the previous subtree call *Call Subtree Arc* is executed, so that when a project number is detected, *Subtree Arc Project* is called instead.

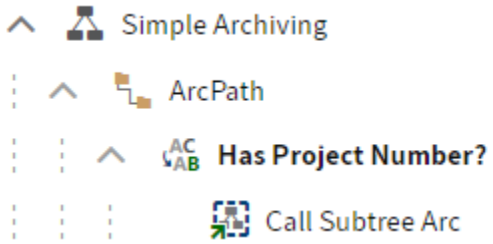
In the *Simple Archiving* active action tree, we now select the *ArcPath* filing path action:



Below this, create the *Match/Replace* action and give it the name `Has Project Number?`. Set the start condition as `true`:



The new action appears between *ArcPath* and *Call Subtree Arc*:



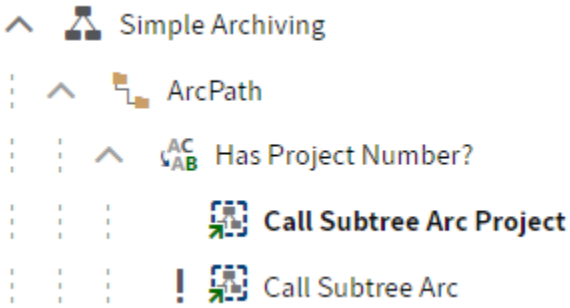
Now create a copy of *Call Subtree Arc* for the reverse case.

Method

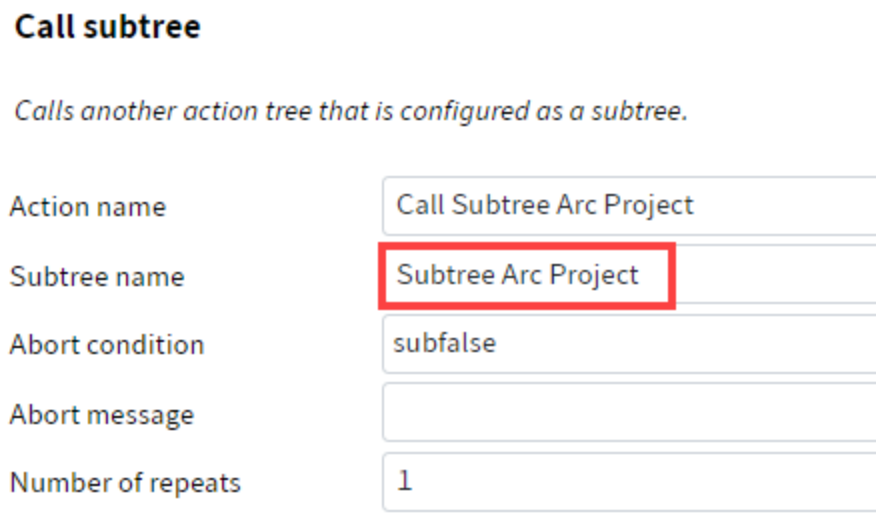
1. Select the *Call Subtree Arc* action.
2. Click *Copy*.
3. Select the ... action.
4. Click *Paste*.

Result

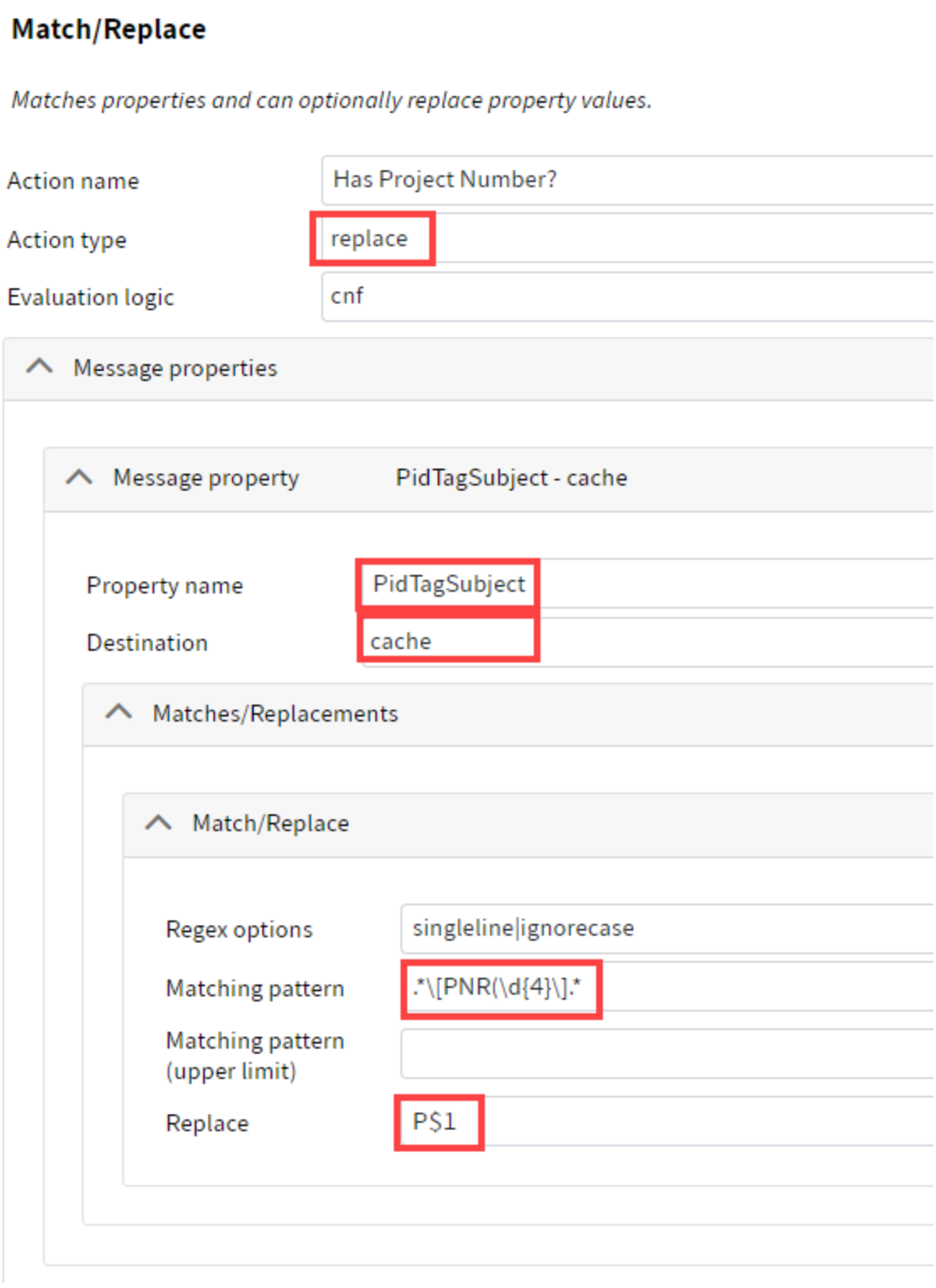
We now have two identical subtree calls. We rename the action for the case *Project number found*:



In the process, *Call Subtree Arc Project* must also call the correct subtree for the project repository:



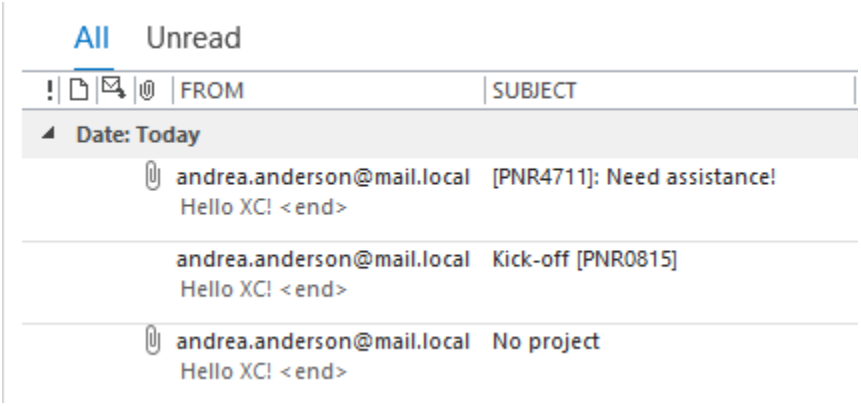
Now that we have arranged the flow of actions logically as required, we need to configure the case distinction in the *Has Project Number?* action:



Since we need to convert from *PNR* to *P*, the action type must be set to *replace*. Otherwise, the value would only be matched but not replaced. The e-mail subject is the *PidTagSubject* property. To ensure that the property value generated by the replace action is retained in the copy of the property, we need to set *cache* in the destination field. We configure the matching pattern with the patterns generated by the regex tool.

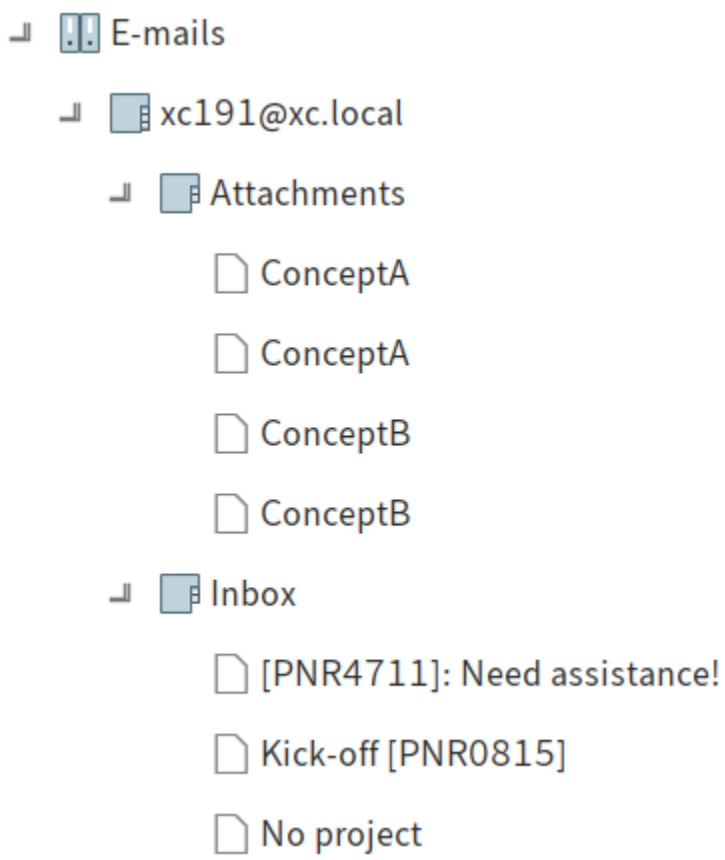
Examples

Once you have saved, validated, and published the configuration, you can generate the first messages:



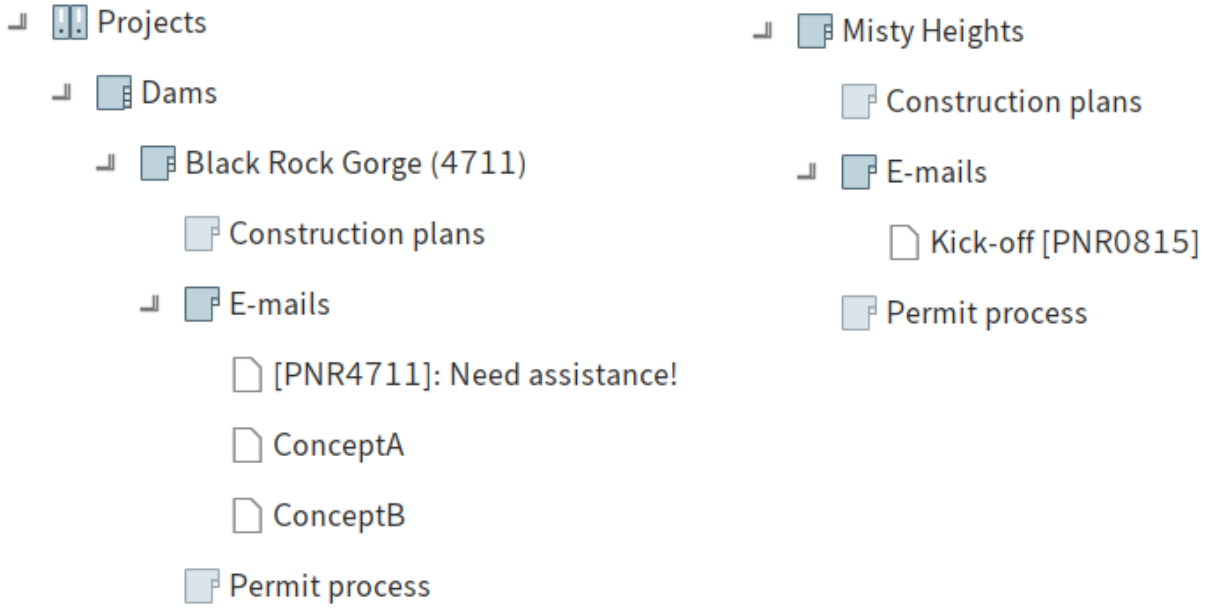
We will use a message without a project reference, but with attachments. To test the dynamic recognition of project numbers functionality, we use two messages with different project numbers according to our definition of project numbers, where one message has attachments and the other does not.

After processing by ELO XC, you should be able to see this result in the repository:

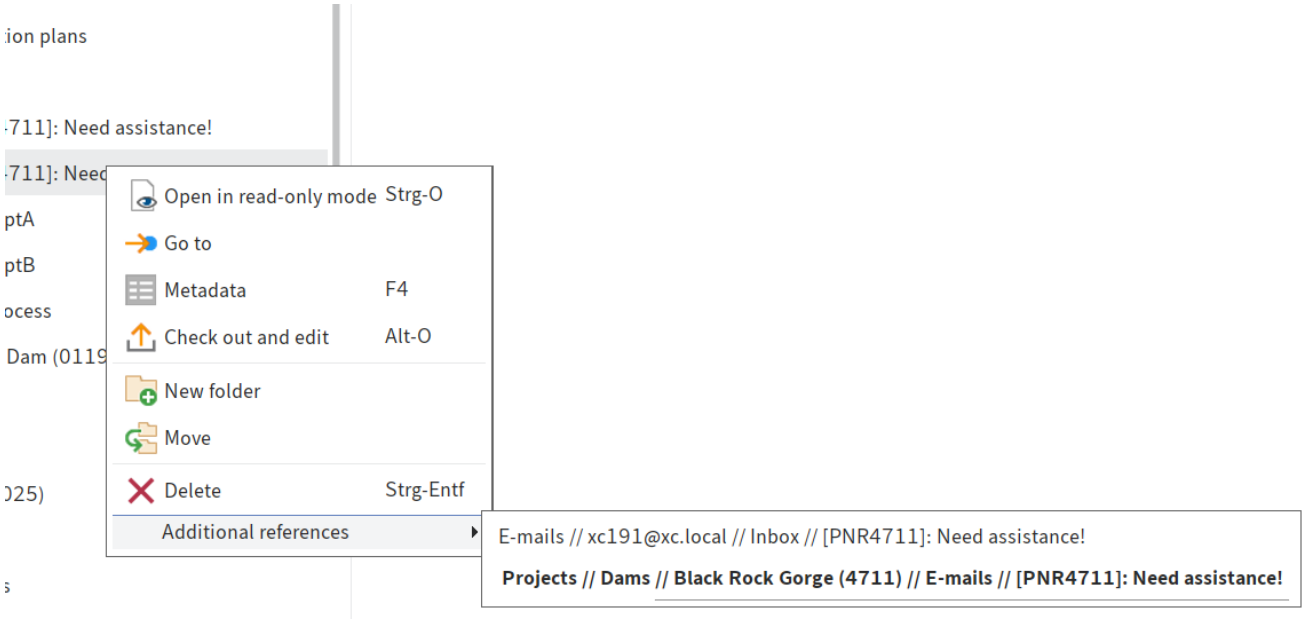


The message parts were stored in the usual way. *No project* and *Need assistance* both had attachments, so four attachments were stored.

Note the new order of the logical references in the project folder structure:



The references were created to match the value in *ELOINDEX*. You can now use the references via the context menu in the client:



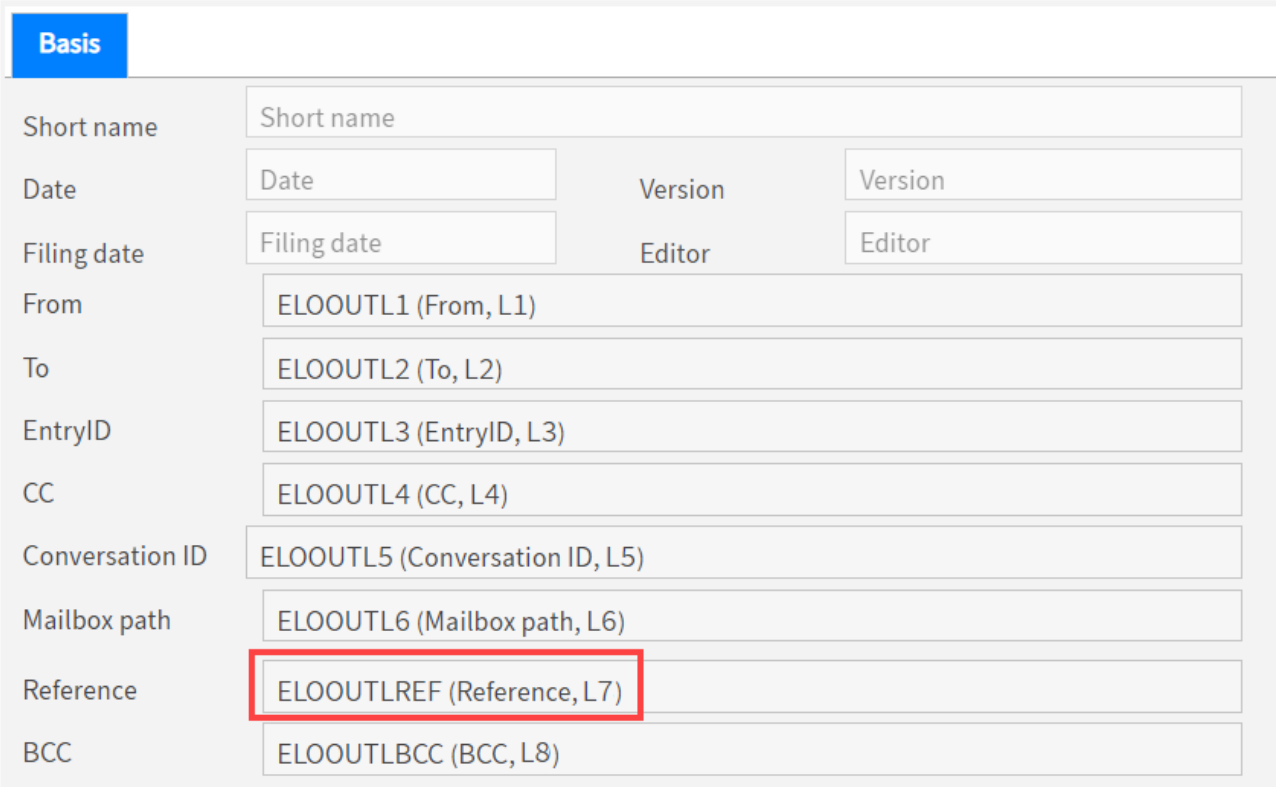
If you want to set the references exclusively for the e-mails or for the attachments, the following types of SORD references are available: *maindoconly* and *attachmentsonly*. The principle is the same. Try some more examples with other e-mails. Vary the project numbers or the number of attachments.

Exercise:

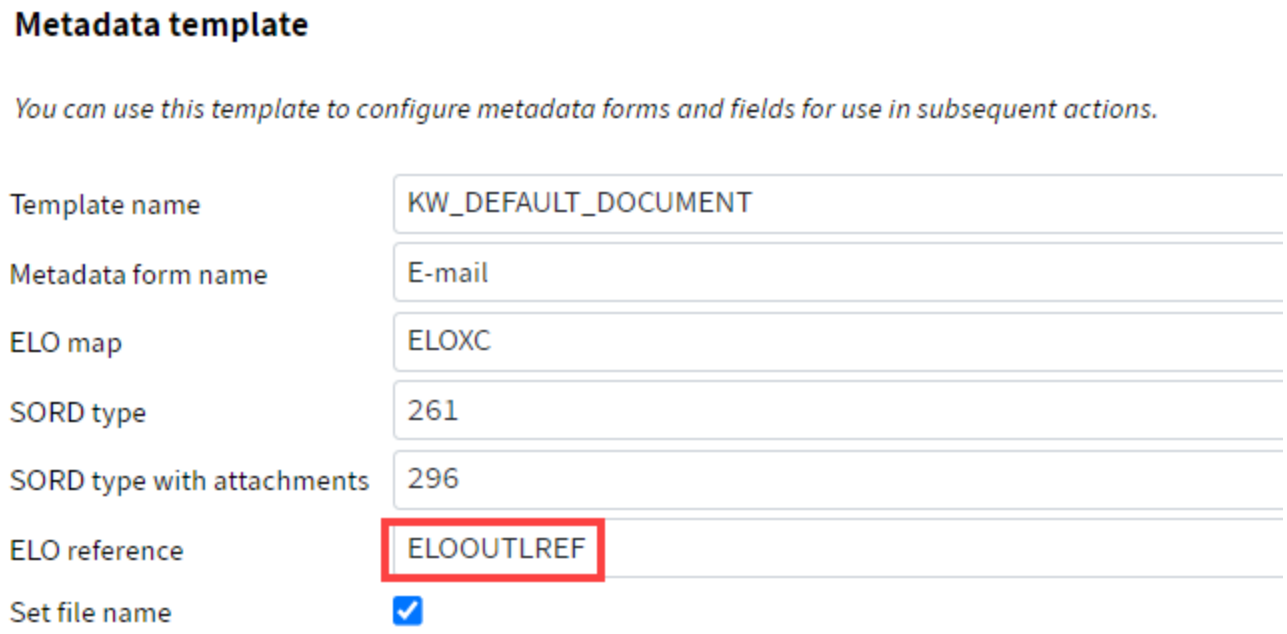
Try to find an alternative minimal configuration of the action trees, because you do not generally need the whole copy of the *Subtree Arc*. You will find the solution at the end of this section.

Implicit link

There is a predefined field group for extracted attachments in the metadata:



The standard e-mail metadata form contains a corresponding metadata field. In ELO XC, the use of this default field group in metadata templates must be explicitly configured:



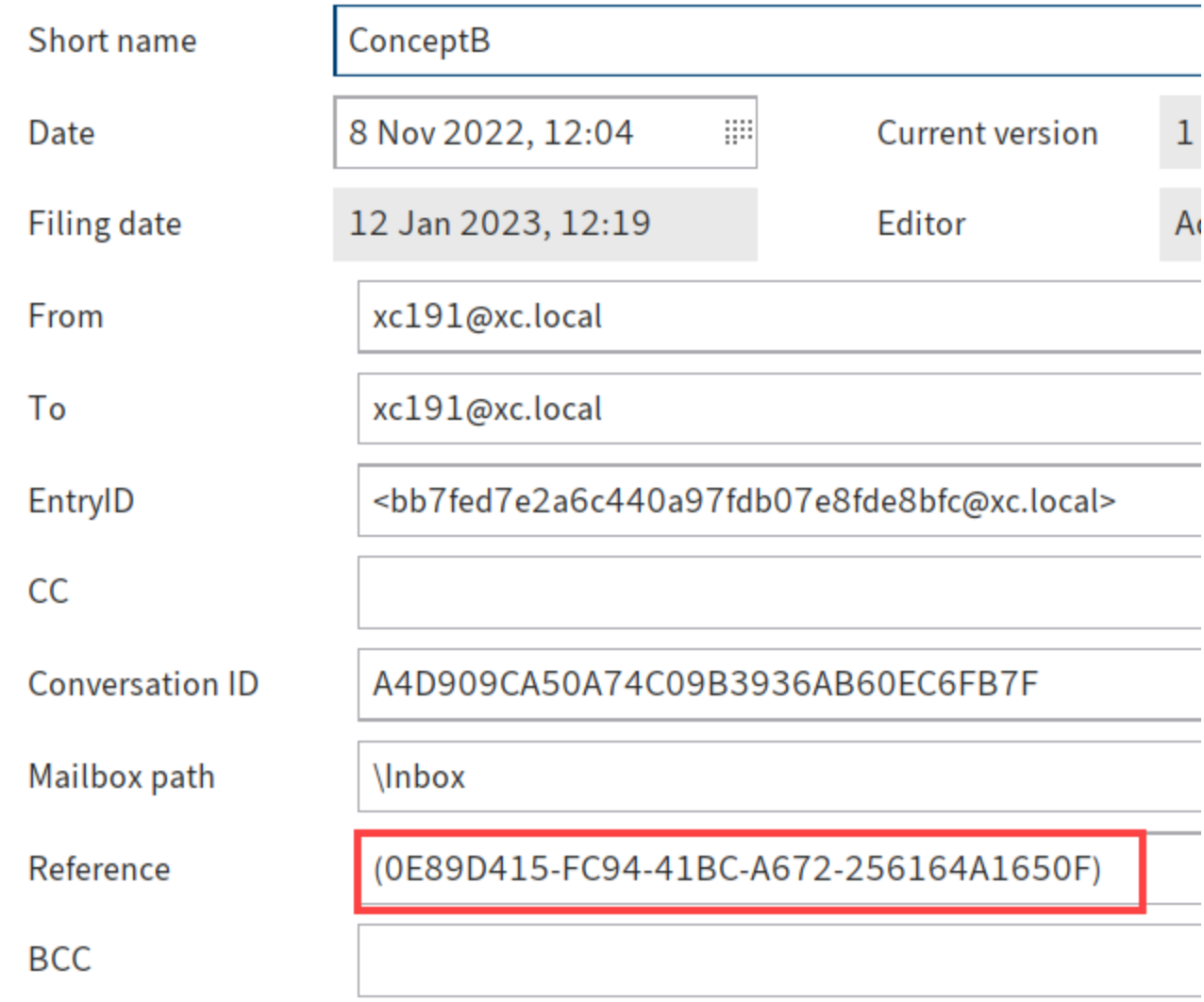
The default metadata template *KW*DEFAULT*DOCUMENT* always uses the ELO reference *ELOOUTLREF*. The *E-mail* metadata form and the default metadata template *KW*DEFAULT*DOCUMENT* should always be used as an example of e-mail metadata. Even custom metadata forms should contain the field groups of the e-mail metadata form.

ELOOUTLREF: The SORD GUID of the original message is always set as the value in the case of extracted attachments. However, this only works if the message that the attachment was extracted from was also archived. If the *attachmentsonly* export type is used, no message/main document is archived, meaning that*ELOOUTLREF* cannot be assigned a value.

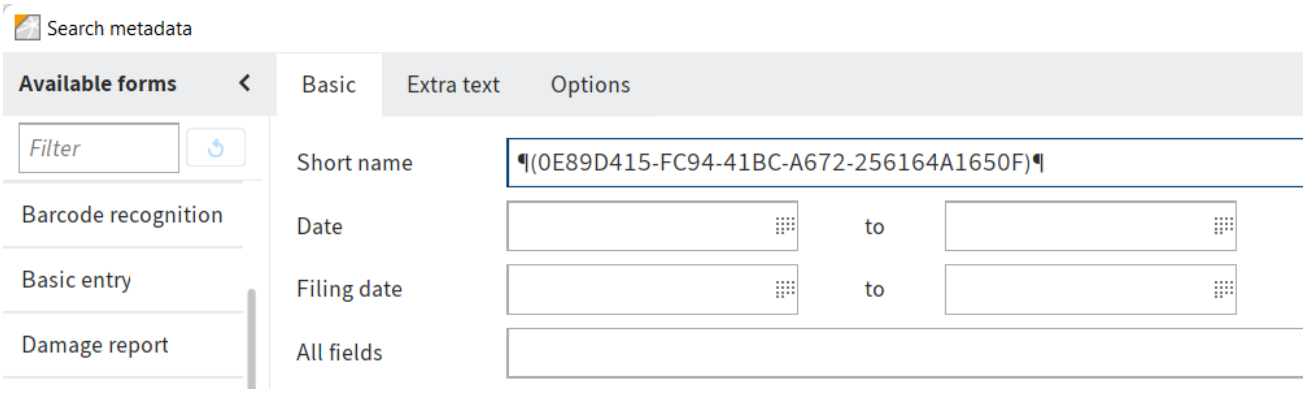
You should consider carefully whether to use it in your own metadata templates. This also means that custom metadata forms used to archive e-mails with extracted attachments should include this default field group.

Example:

Extracted attachments were stored without SORD links. At a later point in time, you want to find the corresponding original message for an attachment. The metadata of the PDF document has an entry for *ELOOUTLREF*:



You can use this SORD GUID in the metadata search:



Summary

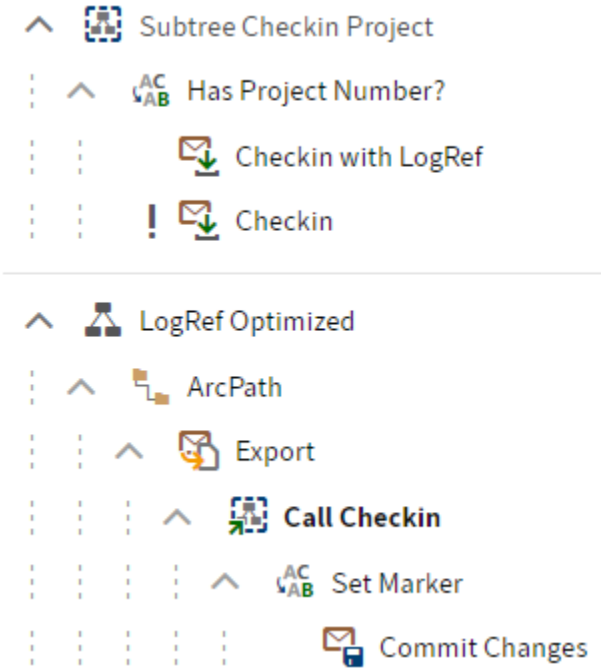
-

SORD links are useful for associating separated message parts with each other permanently, regardless of where they are stored in the repository.

- Logical references can be created just by configuring an associated storage path. This simple method assumes that the paths for references are known and can be configured directly.
- With the advanced method for creating logical references, on the other hand, it must be possible to determine reference paths by comparing a message property to an existing metadata field value.
- The use of the reserved field group *ELOOUTLREF* is intended for matching file attachments to their original messages. Custom metadata forms should use this field group accordingly.

Solution to 'Minimal configuration exercise' in the 'References by metadata field' section

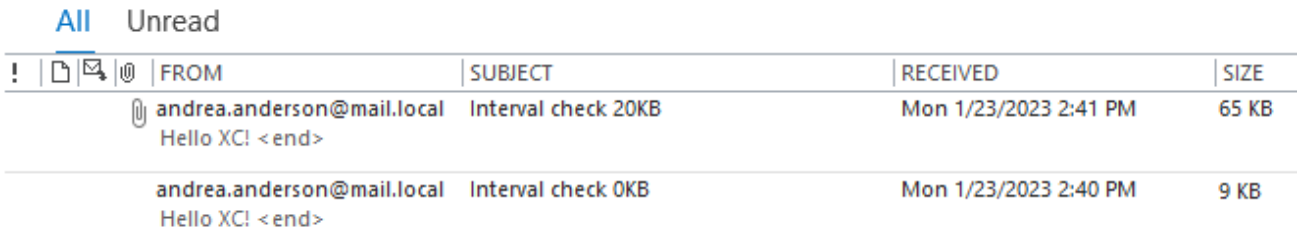
Since the distinction is made by determining the project number and is only required during the filing process, only the *Archive* action is moved to a subtree for the different cases:



The *Call Checkin* subtree action calls *Subtree Checkin Project*, because the case distinction is only needed for the *Archive* action.

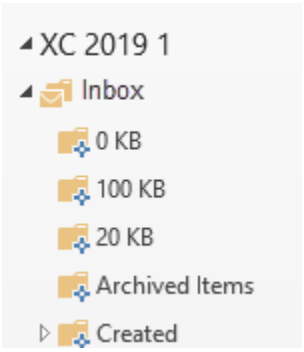
Interval check

The *Match/Replace* action in ELO XC is used to evaluate property values. The action uses regular expressions, which help to recognize and modify complex string patterns. Checking whether a property value is within a specified interval constitutes a special case. Using regex expressions for this is a complicated method that has its limitations. You therefore have the option to configure *matches* that allow you to set interval thresholds.

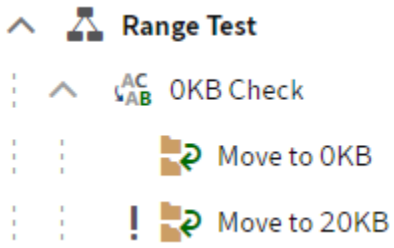


Let us assume that you only want an action tree to process messages that are larger than 20 KB or 20000 bytes. In a simple use case, you would use the action tree selection restrictions, but in this example we will assume that we are not creating any number of action trees and that ELO XC should perform different cascading size checks. Setting the limit at 20 KB would mean that 100 KB, 500 KB, and 1 MB are also included, which would lead to a more complex series of checks. Each case distinction would require different follow-up actions.

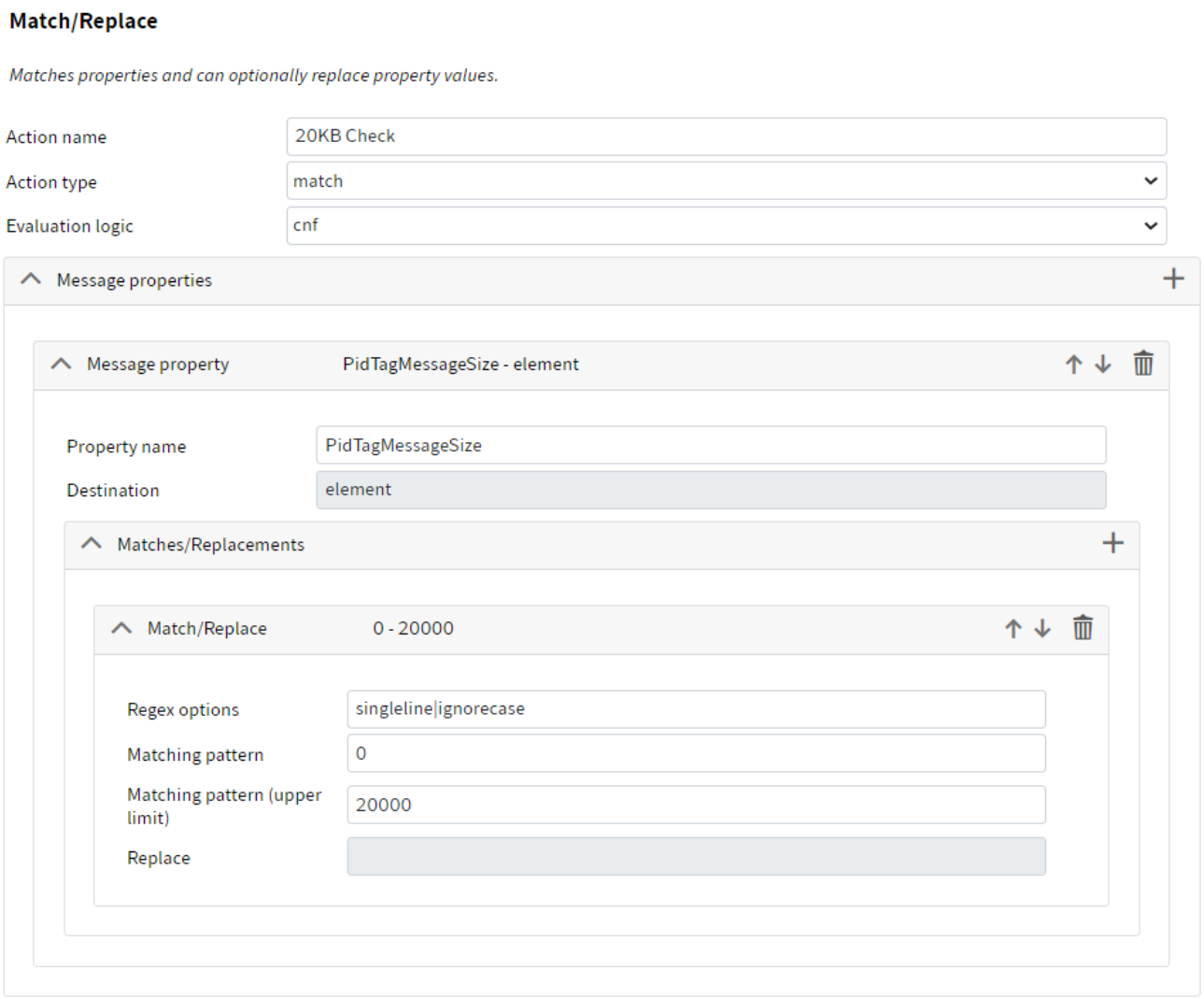
To illustrate the interval check, this example also uses the *Move* action, which moves the corresponding messages to the mailbox folders */Inbox/0 KB* and */Inbox/20 KB* on a case-by-case basis. The folders should only contain messages that have the required minimum size.



The check and the change are clearly illustrated in the action tree.

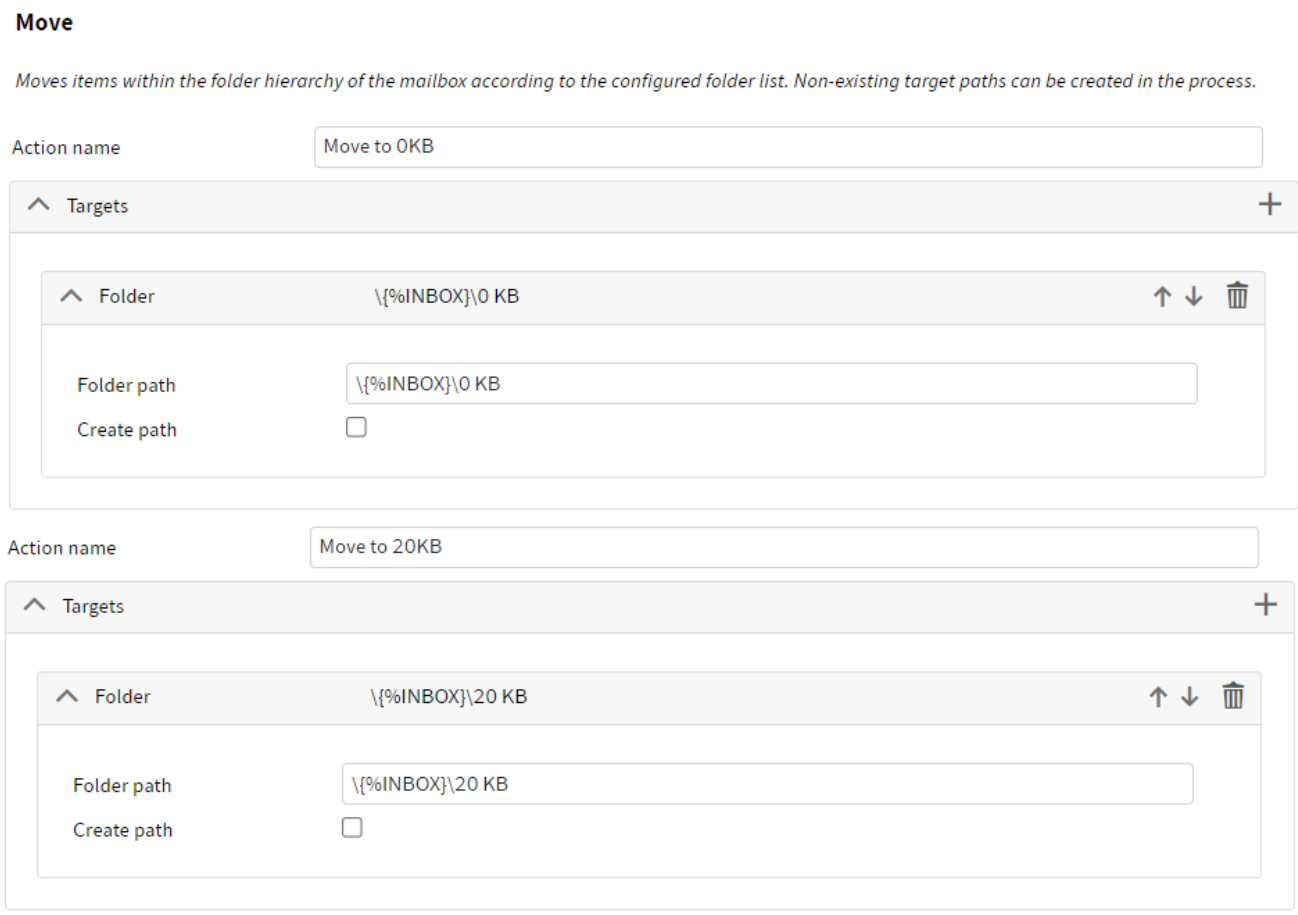


The action tree only processes messages that have not been archived. The actions are configured directly. The *Match/Replace* action is required for this example.



If you process the two messages with this action tree, the messages will be moved to the correct folders.

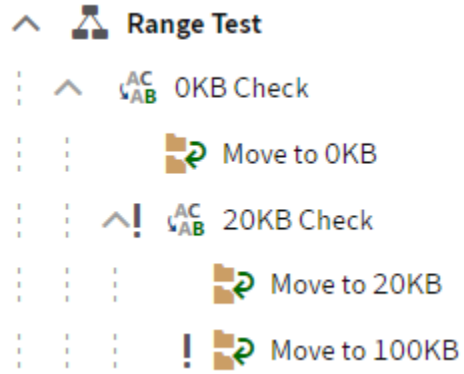
In a real-life scenario, you would set up additional cascading size checks in this way:



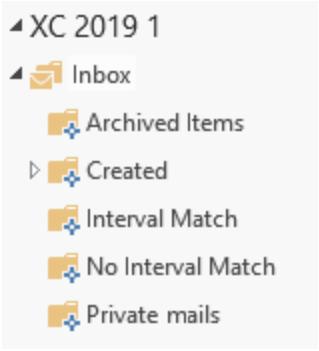
You can also use this method to check other properties. The underlying data type is important for the interval check. You cannot perform an interval check on *strings*. Only numeric types (*integer*, *long*, *float*, *double*) and calendar data (*DateTime*) are suitable for this type of check.

Let's assume you need to pre-sort messages according to the time of delivery. ELO XC uses the delivery time *PidTagMessageDeliveryTime (type DateTime)* as the default property for determining the age of an item.

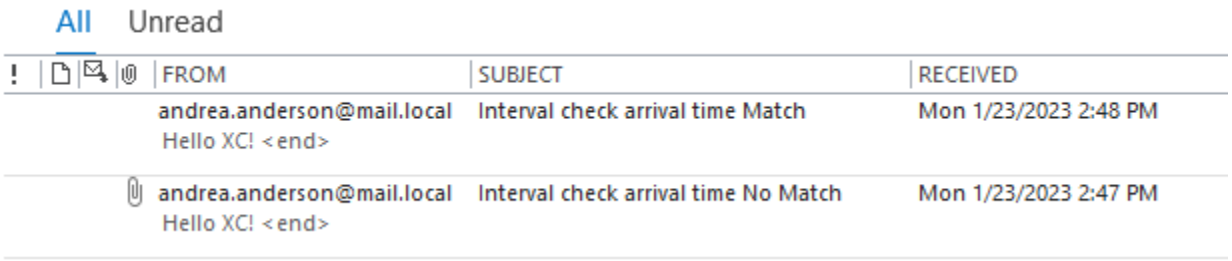
The following examples illustrate how to configure the above check for calendar data. For the check by delivery time, the target folders in the mailbox are renamed.



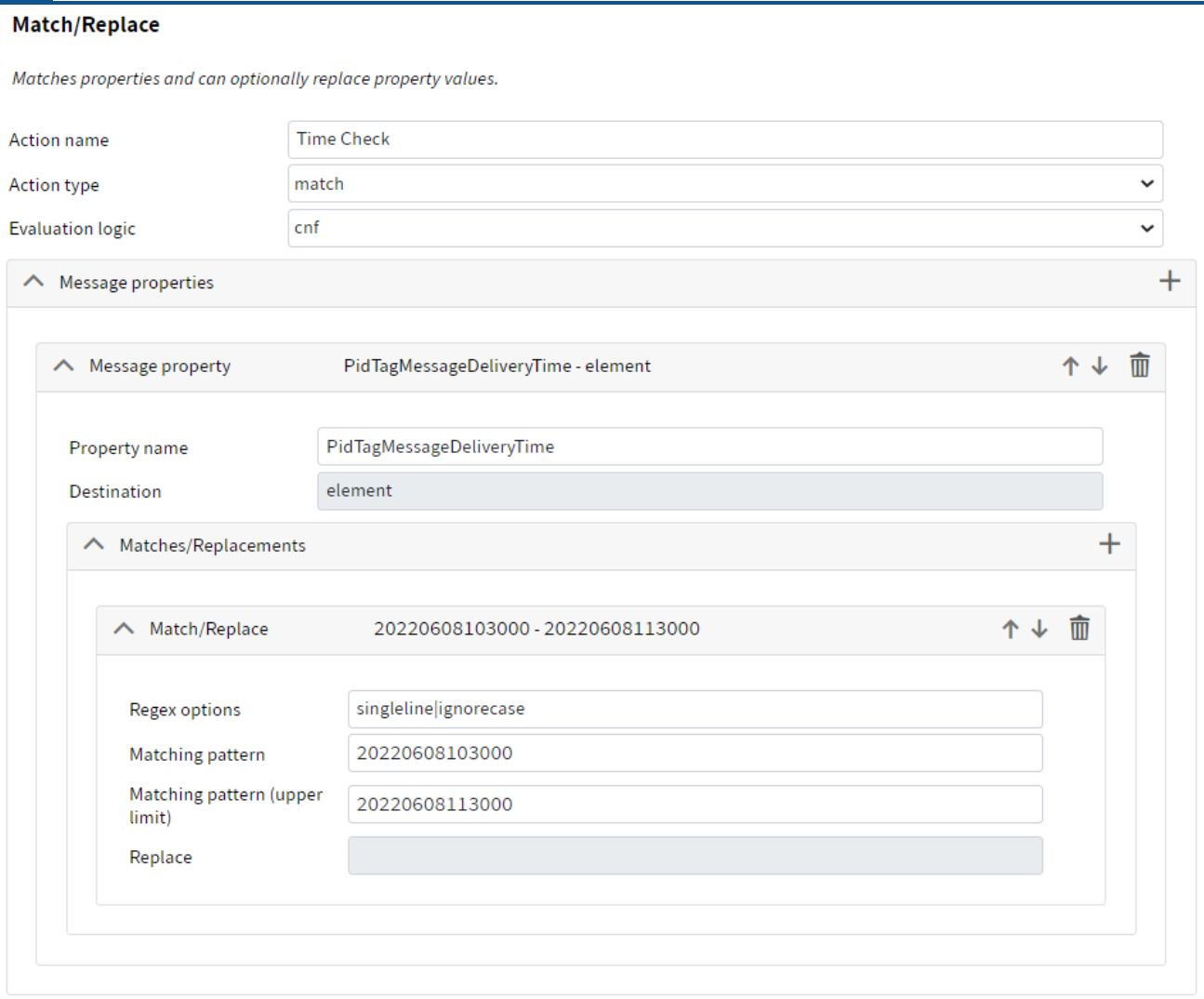
Interval Match and *No Interval Match* are the target folders. The action tree has the same structure but has a different name because of the different intended result.



The test messages should have completely different delivery times.



The positive check should be configured so that the match confirms the interval on June 8, 2022, between 12:30 and 13:30.



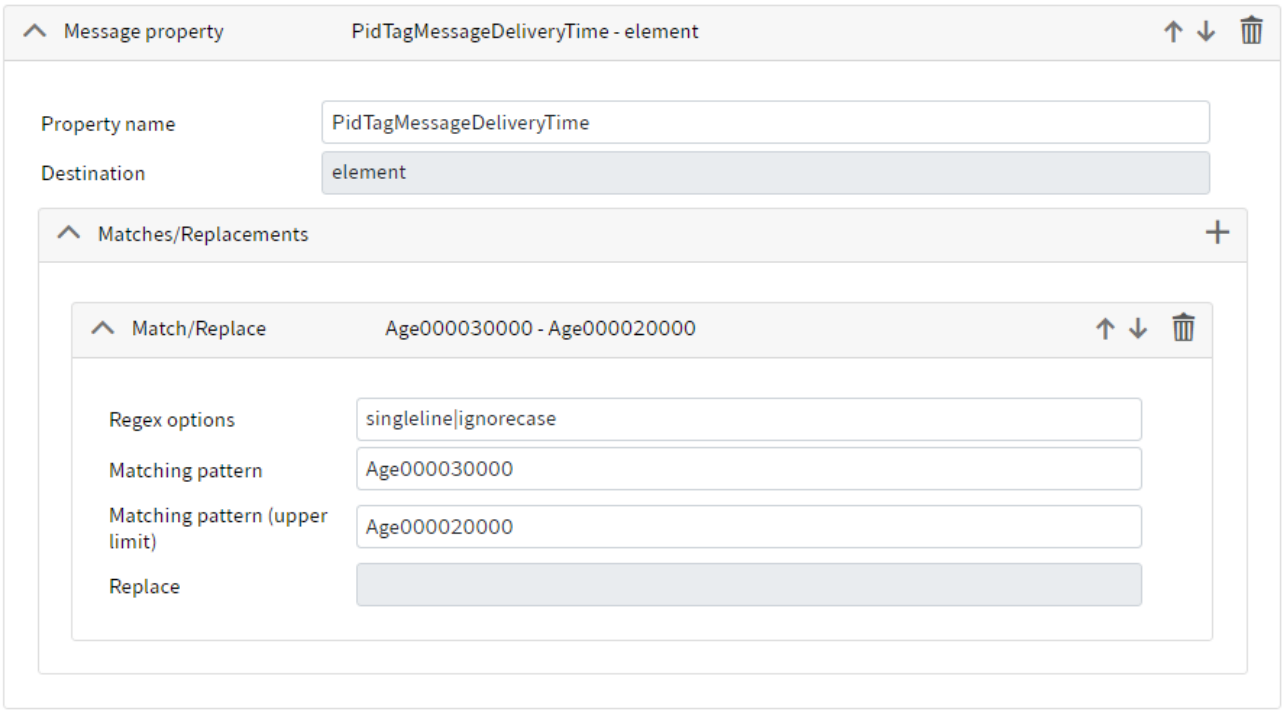
Besides entering the correct property name, you must also select the upper and lower interval thresholds according to the date type. This requires the *DateTime* internal representation type. This type always has the ISO date format (14 digits in the order year, month, day, hour, minute, second). There should be no digits missing.

In this example, a time between 12:30 and 13:30 is relevant. However, the configuration uses `20220608103000` and `20220608113000` because all properties with timestamps are normalized to UTC time. Only Outlook performs an automatic localization of the timestamps and changes the view according to the set time zone.

Let's assume you now want to check the message age instead of the interval with a fixed time. Static timestamps are not suitable for this. You can configure this by changing the matching pattern, combining the prefix *Age* (case-sensitive) with an age specification. An age specification always has the format *DDHHMMSS* with *D* for the days, *H* for the hours, *M* for the minutes, and *S* for the seconds.

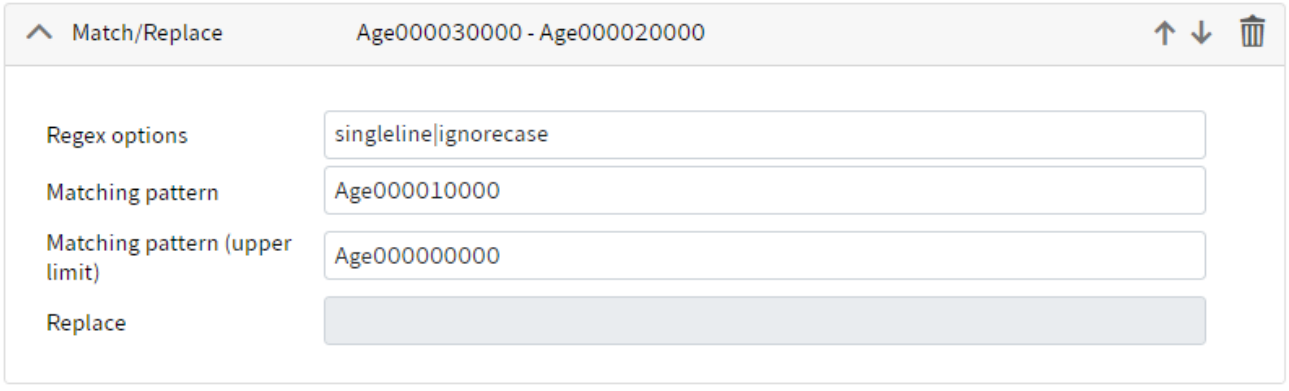
If you want to specify an age between four and five hours, you would use the interval of the *Age000050000* and *Age000040000* thresholds. The lower threshold should represent the higher age.

If you want to process the two messages on June 8, 2022 at around 15:00, these two matching patterns are sufficient:



When determining the message age, i.e. the time difference between an imaginary now and the delivery time of a message, the time of calculation is especially important, since a message ages from the time of its delivery. The reference point for calculating the age is therefore the internal timestamp, which is always generated at the beginning. Although a job can take longer, these uncontrollable dynamic factors are eliminated by using the main job timestamp (in *EloXcArchivedBase* or *EloTimeStampJob*) to apply a static reference point for age calculation.

The interval thresholds with age calculation can be modified even further. Let's assume that only messages that are no older than one hour since the job start are relevant. In this case, you would set the lower threshold to `Age000010000` and the upper one to `Age0000000`.

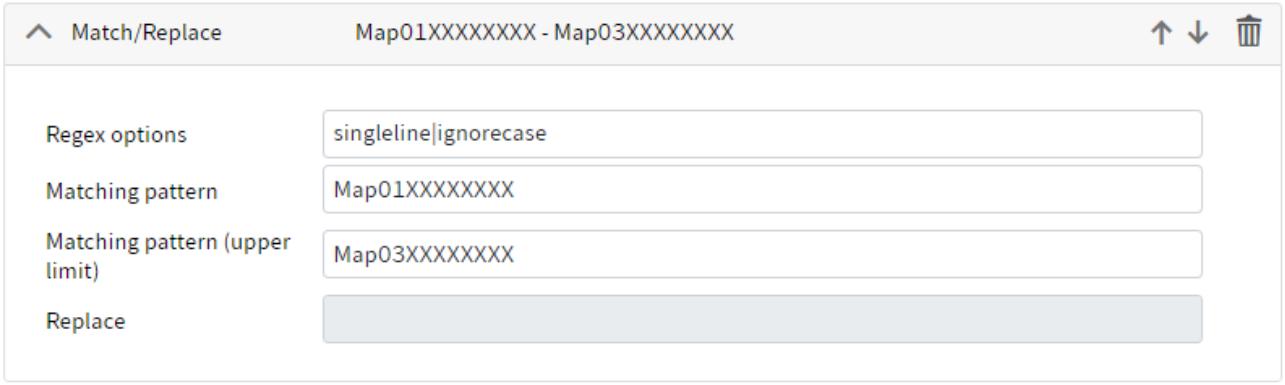


The age specification `000000000` (nine zeros) tells ELO XC to choose the available extreme values. The upper interval threshold is therefore automatically the job start, since the latencies since the

start are technically eliminated. The lower threshold is `01.01.1970 00:00:00`, a definite date in the past. Both extreme values are hardwired.

The specification of static time limits in ISO format when entering `0000000000` (14 zeros) also uses two extreme values. If zero is the lower threshold, `01/01/1970` is used internally as with age limits. However, if zero is the upper threshold, it is converted to the timestamp at the time the match occurred during processing.

There is a third way to perform checks with time intervals. Let's assume that you want to filter only those messages whose delivery time is within a specific range of months, days, hours and other date parts. For example, you want to process all messages of the first quarter.



The pattern has the *Map* prefix. The entries are case-sensitive. The prefix is followed by ten digits for the date, two each for month, day, hour, minute, and second. Years are not allowed in this format. An `X` means that a date part of the pattern, such as day or second, is completely discarded, and is instead inherited from the property value. If a date part is represented by digits, these are inherited from the matching pattern. The different date parts in the pattern, except for the year, are combined with the property value, resulting in the two interval thresholds.

The first quarter consists of months 1, 2, and 3. In ISO format, months have two digits, i.e. 01, 02, and 03. Suppose you needed to filter delivery times between 8:00 and 12:00 UTC. The interval thresholds in this case would be `MapXXXX08XXXX` and `MapXXXX12XXXX`.

If you want to filter times from the third quarter to the first of the month between 16:00 and 16:10, you need `Map07011600XX` and `Map09011610XX`.

User import

Preparations

ELO users and their access rights in the repository are central to e-mail storage because correspondence, business or otherwise, should not generally be compromised. It is possible to prepare the repository manually before archiving any e-mails, or it can be done automatically by ELO XC. Central to e-mail archiving is the external mailbox owner whose messages are to be processed and later made available in the repository. ELO XC groups all external mailbox owners in mailbox catalogs, which are created when external directories are queried.

Automatic import of user accounts from the catalog is not a default setting in ELO XC, because it can lead to conflicts with the LDAP import function in the ELO Administration Console or existing legacy data. However, if you have already used the ELO Indexserver LDAP connection, there is nothing to prevent you from allowing ELO XC to import users, but you should consider whether this is actually necessary or whether the LDAP connection is not the better function. ELO XC is limited to mailbox properties. Nevertheless, both import methods are compatible as they both identify the external users based on the GUID in the directories. This allows ELO XC to import user data before the ELO Indexserver LDAP connection is used without preventing it being used later on.

With local directories, the integrated ELO XC mode is generally used for the LDAP connection, which delegates the import to the Indexserver and guarantees uniform data transfer.

The automatic import of user accounts therefore requires careful planning, which should essentially be based on which interfaces are available, at what time user access is to be expected, and to what extent the local administration can independently transfer user data into ELO.

In addition, the different mailbox types processed by ELO XC as well as the filing path influence the resulting permission settings after an e-mail has been stored. Therefore, this usage example consists of several parts, which build on one another and also illustrate central control mechanisms of ELO access permissions.

We recommend that you test the configuration examples and verify the usage example with your own tests depending on your individual level of experience.

You need:

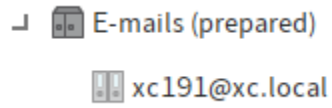
- ELO 21
- ELO Administration Console
- ELO client
- Microsoft Outlook
- 3x user mailboxes – 1x sender, 2x recipients
- 1x shared mailbox (with these user mailboxes as delegates)
- ELO XC
- Microsoft Active Directory
- Microsoft 365
- Remote PowerShell (PowerShell with EXO module 2.0.4 or higher)
-

App registration with certificate authorization in Microsoft 365 tenant and app permissions for delegation via EWS and Exchange management in the role of Exchange administrator

Standard method

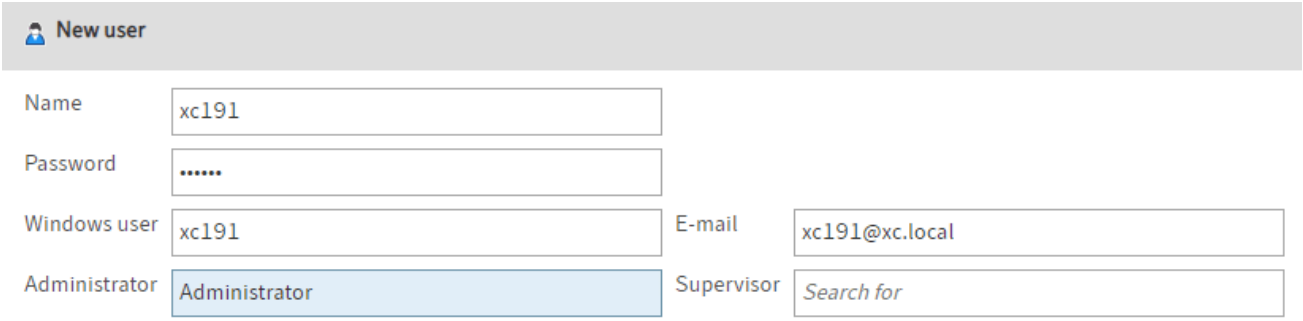
The standard method in ELO XC is based on the assumption that the user structure has been configured in the repository before the first mailboxes are archived. Mailboxes are only associated with ELO data via the filing path using the automated functions of form permissions, which also play an essential role in the following examples with automatic user import. The standard method is used specifically if you want to separate external users and ELO users. However, it must be used if a static mailbox catalog has been configured.

In this example, a user gets a separate folder in the main folder *E-mails (prepared)*, in which messages will be stored according to the folder structure in Outlook. This folder should be named after its SMTP address:

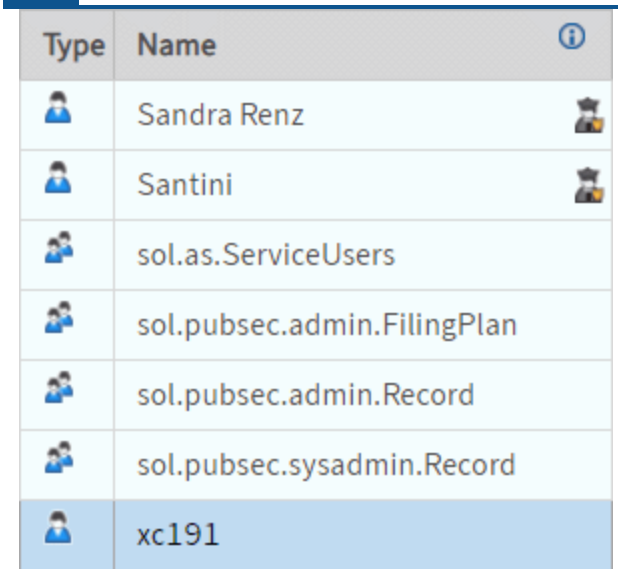


Both folders are created using the default folder form so that all users have access to them. However, since we want to configure a user-specific e-mail repository, the stored e-mails must be protected against unauthorized access by other users using appropriate permission settings.

The first step is to create the user. This is done in the ELO Administration Console:

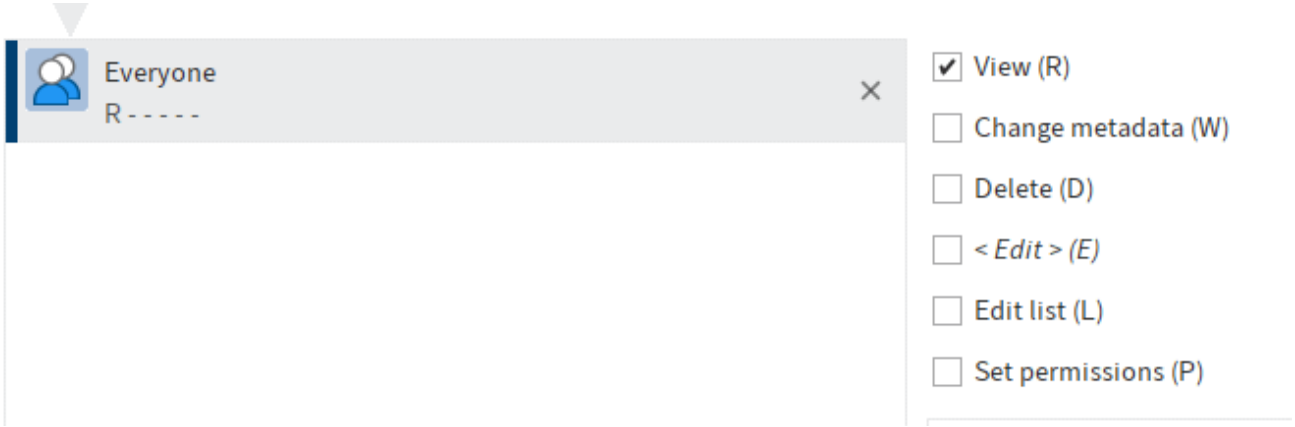


After you save the new user, it shows up in the list:

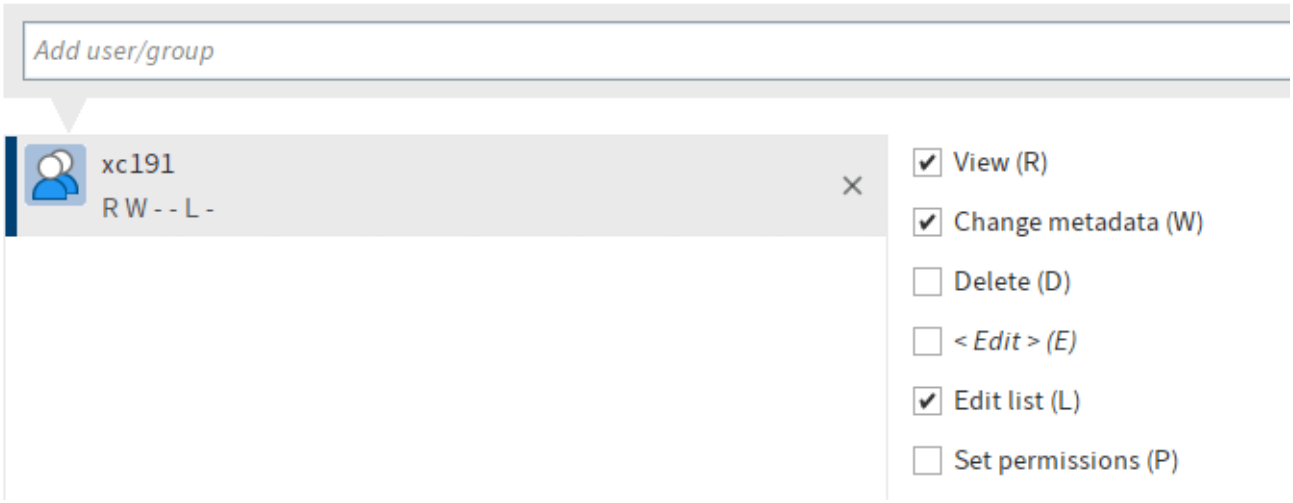


In the client, you now set the permissions for both folders. Navigate to the main folder *E-mails (prepared)*.

All users have read access to the main folder:

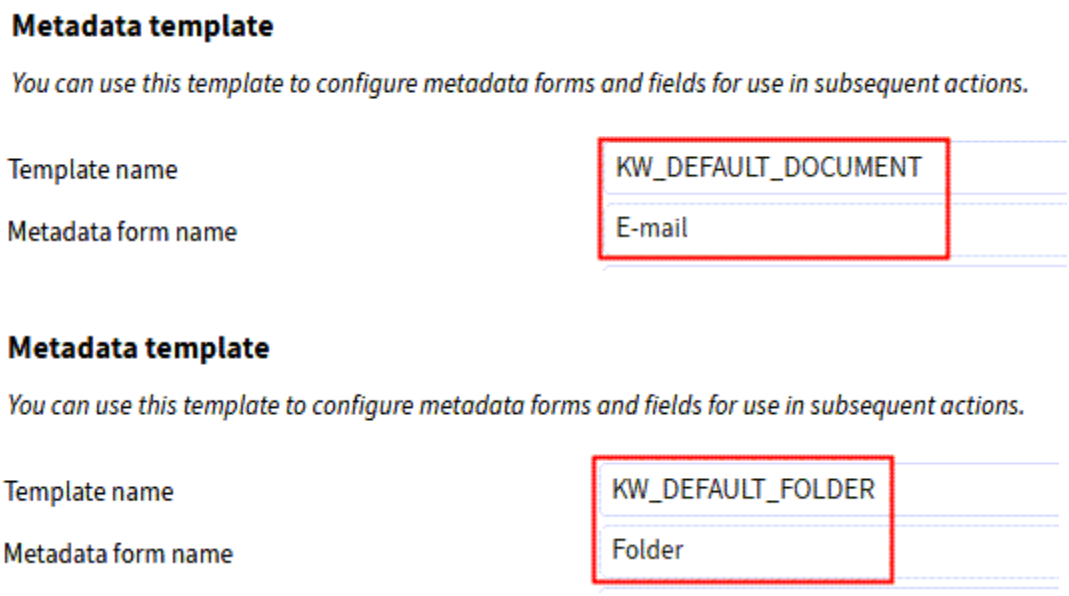


The user folder that will contain the stored mailbox is assigned permission settings that grant *xc191* the rights to read the e-mails, move them, and change their metadata as needed:



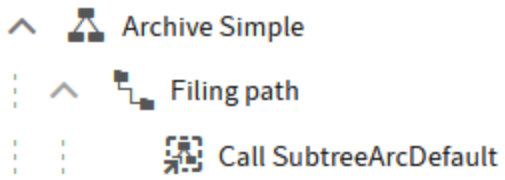
Once the folders have been configured accordingly, messages for *xc191* can be stored. The expected Outlook folder structure is automatically secured based on the permission settings of the metadata forms used. The default *Folder* and *E-mail* forms already have the required *Parent rights* permissions settings, which ensures that permissions for the transferred mailbox folders and messages are passed on correctly during processing.

We will now use ELO XC to configure simple e-mail storage. The action is based on the metadata templates *KW*DEFAULT*DOCUMENT* and *KW*DEFAULT*FOLDER*. *KW*DEFAULT*DOCUMENT* is assigned to the stored documents or messages and *KW*DEFAULT*FOLDER* is assigned to the created folders:

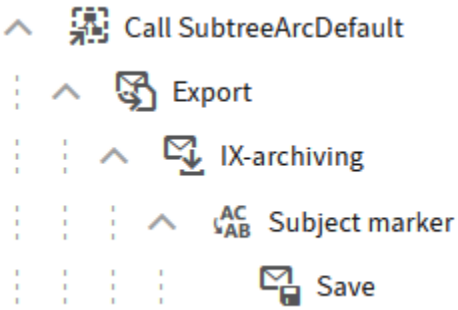


Both metadata templates are part of the default settings of ELO XC, which is why they are verified every time ELO XC is started. If they are missing, ELO XC automatically creates them. In addition, they already contain the default mappings of message properties to metadata fields, which is why they shouldn't be changed. If other templates are required, you can create customized copies of the two standard templates.

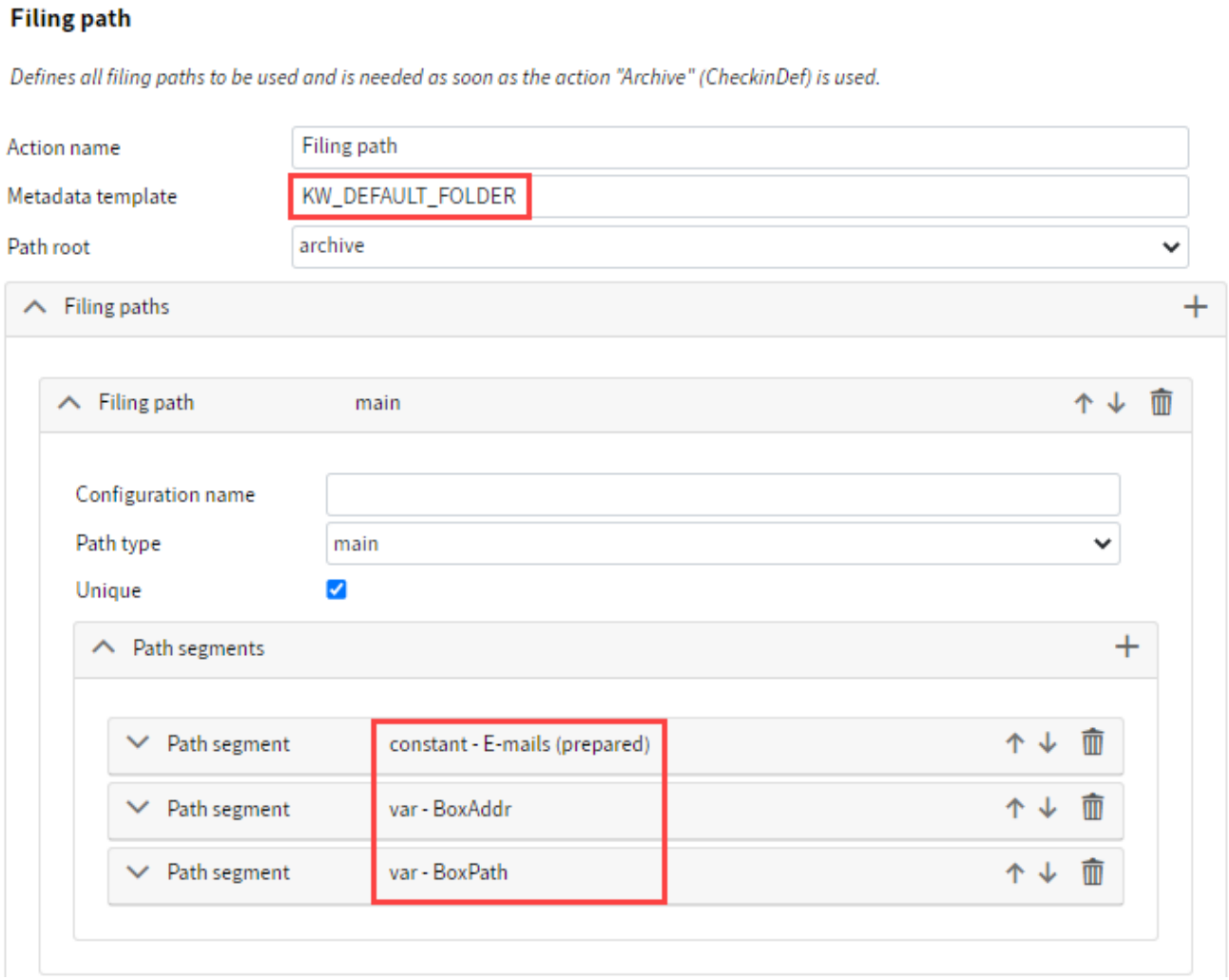
This simple storage process uses an active *Archive Simple* action tree.



The active action tree calls a *subtree* (or template tree) in which *Export*, *Archiving*, *Tag*, and *Save* are configured.

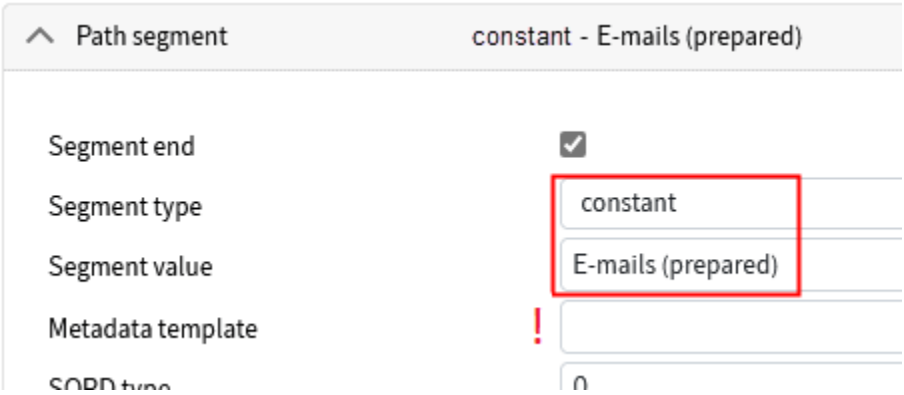


To ensure that the correct filing target is used in the repository, the *Filing path* action must be configured in advance.



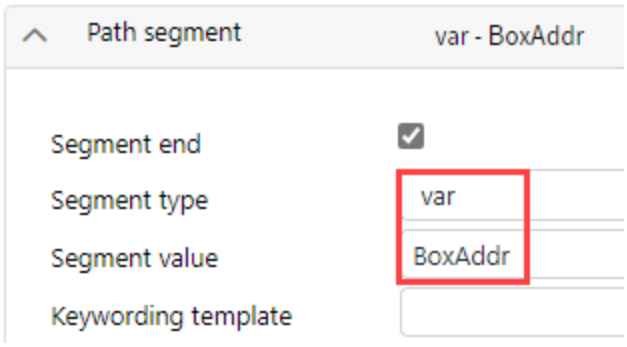
During filing, the path is created by using existing path items and creating missing ones. This ensures that the settings of the folders *E-mails (prepared)* and *xc191@xc.local* remain unchanged. This also means that the path segments in the repository are not changed by ELO XC once they have been created because the only identifier used is the folder name.

The segment for the main folder is configured as follows:

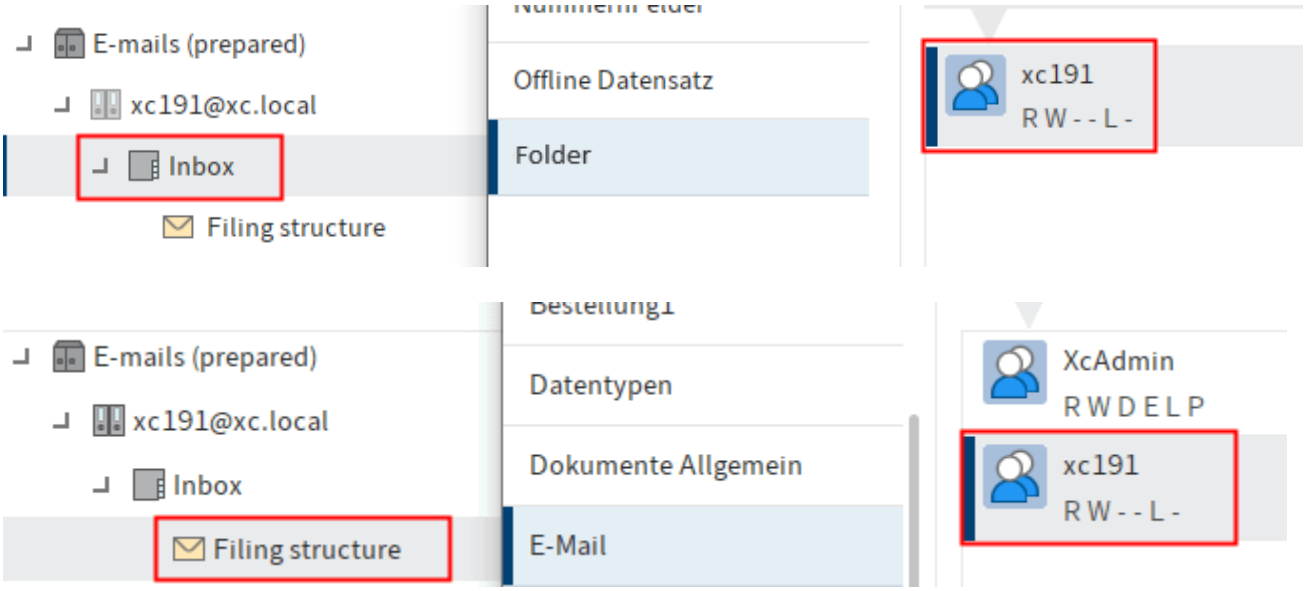


Since the name *E-mails (prepared)* is assigned by default and cannot be changed, the segment type must be set to *constant*. The metadata template of this path segment remains empty because the *KW*DEFAULT*FOLDER* template is already set at the top level of the entire action. If you want to use different templates for individual segments of the filing path, using different metadata forms, for example, you can configure this in this field of the segment. However, the segments are not linked to the metadata in this example.

The automatically generated name of the next segment in this example is the mailbox SMTP address. The variable *BoxAddr* is responsible in this case.



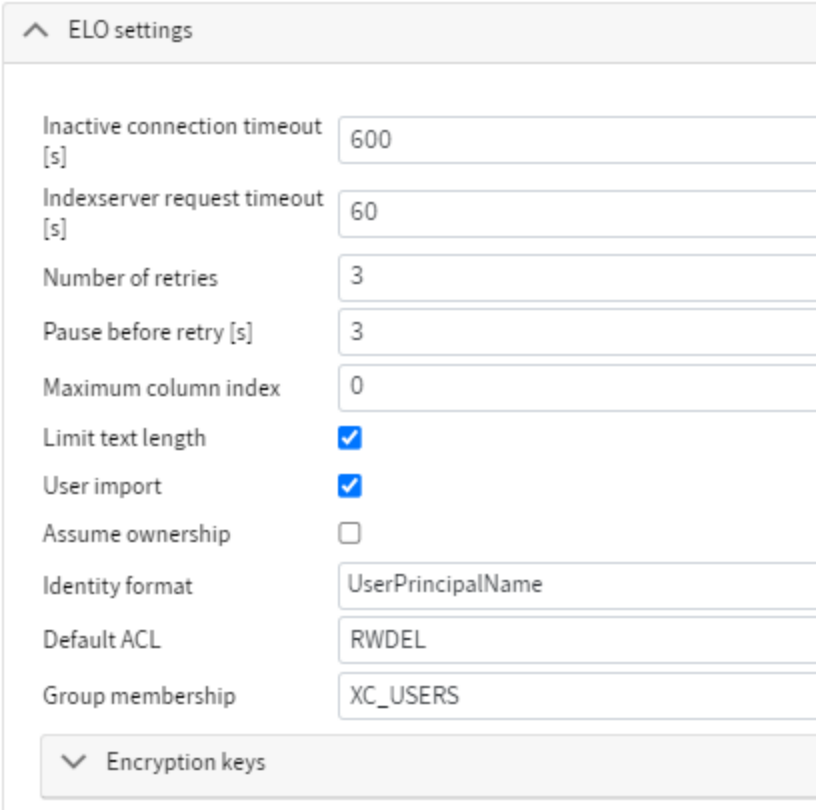
The third path segment uses the *BoxPath* variable, which is used to transfer the structure from Outlook during filing. Now, when messages are processed, you can see in the repository that the desired permission settings are automatically applied correctly:



Automatic import of user accounts

This example with automatic user import for e-mail storage also assumes that there is a main folder, which will now be called *E-mails (unprepared)*. This is where the mailbox folders with Outlook structures will be stored after filing. This does not require any preparations in the repository. Also, no ELO users are created. If you only want to test the configuration, the created ELO users have to be deleted again before each test.

To do this, you need to enable to the automatic import option under *ELO settings* in the instance configuration.



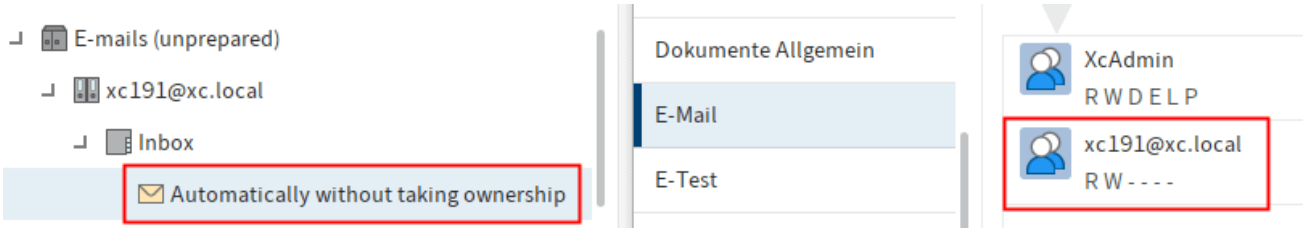
Description of several parameters:

- Assume ownership: Enabling the *Assume ownership* option means that the user automatically created in the repository is also entered as the owner of the newly created documents.
- Identity format: This selection field determines which catalog property to use as the ELO user name. The possible properties are *UserPrincipalName*, *sAMAccountName*, *CN*, and *IxLdap*.
- Default ACL: This where you specify the ACL for the user folder. This permission setting is applied to all transferred items based on the inheritance mechanisms of the metadata form. *RW* was selected here to distinguish it from *RWDELP*.

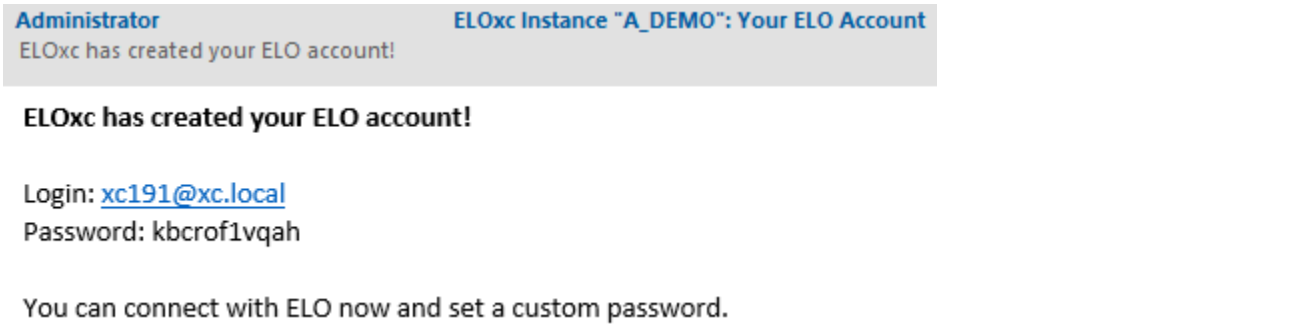
Information

Identity formats: When using *IxLdap*, you must have configured and enabled the Indexserver LDAP connection. In this case, ELO XC does not perform import the users itself, but instead registers the user in the background during processing, which causes the LDAP connection to import the data. When using the other identity formats, ELO XC maps the external user data itself. In this case, the passwords of newly created users are automatically generated and sent to the mailbox owners. This requires a valid SMTP address of the service account used to send the service messages.

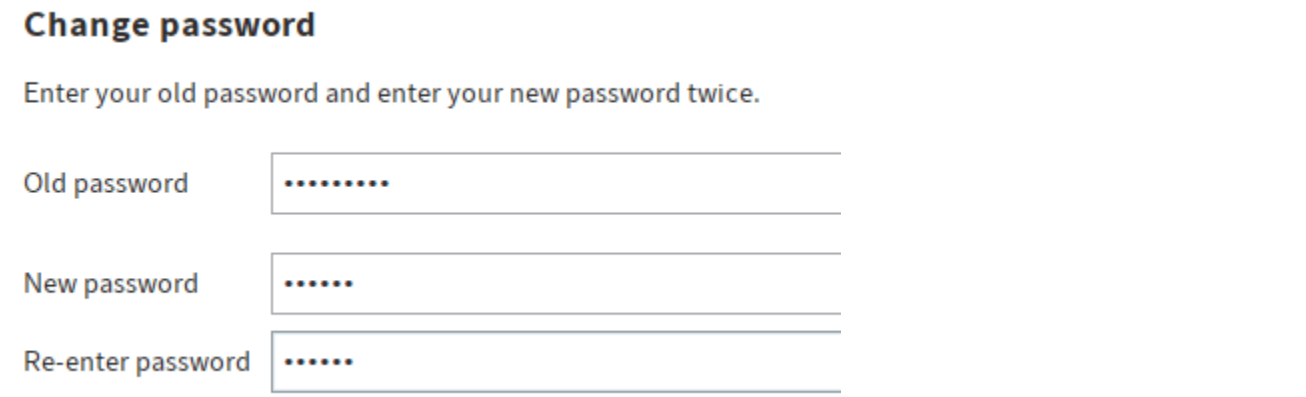
If you store an e-mail with these settings, you will get the following result:



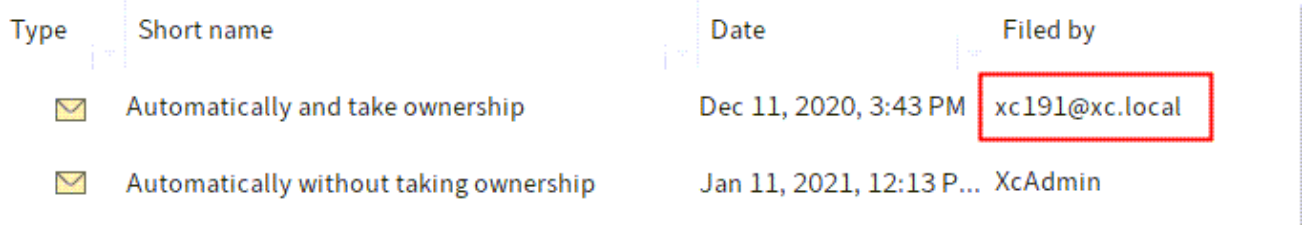
The message *Automatically without taking ownership* is archived and is assigned the configured permissions settings *RW*. In addition, the user *xc191* is notified that the ELO user account was created:



The user is now able to to log on to the repository where they will be prompted to set their own password:

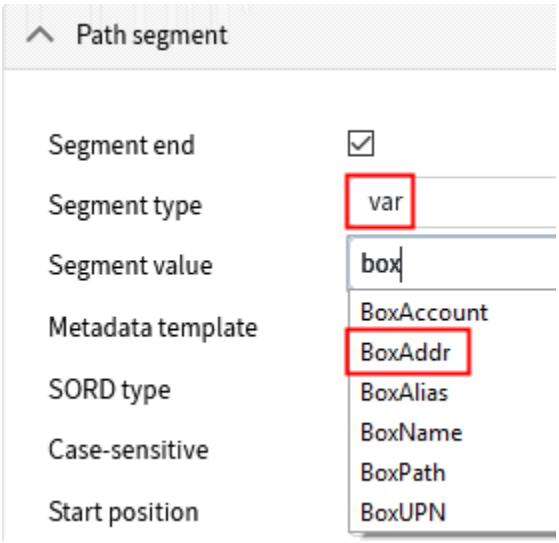


If you enable the *Assume ownership* option in the instance configuration, each stored e-mail and each filing path folder created in this way will be assigned to the mailbox owner as if they had stored it manually:



If you do not enable this option, the stored message will be automatically assigned to the service account.

The assignment of rights in automatic user import only works properly if the filing path uses a variable that is associated with the mailbox owner:



The variables are assigned as follows:

Variable	LDAP user property
BoxAccount	sAMAccountName
BoxAddr	mail
BoxName	displayName
BoxUPN	UserPrincipalName

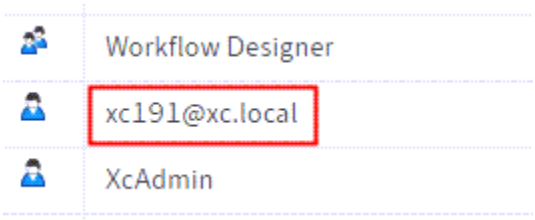
Please note

If one of these variables is detected in the filing path and assigned a corresponding value, ELO XC simultaneously sets the configured ACL for the path segment. It therefore makes sense to use only one variable that is associated with a mailbox for each filing path. At the same time, one of these variables must always be configured for successful and secure mailbox emulation. If this is not the case, data could be compromised. You must configure ELO XC so that it is not possible to create overlapping paths for different mailboxes.

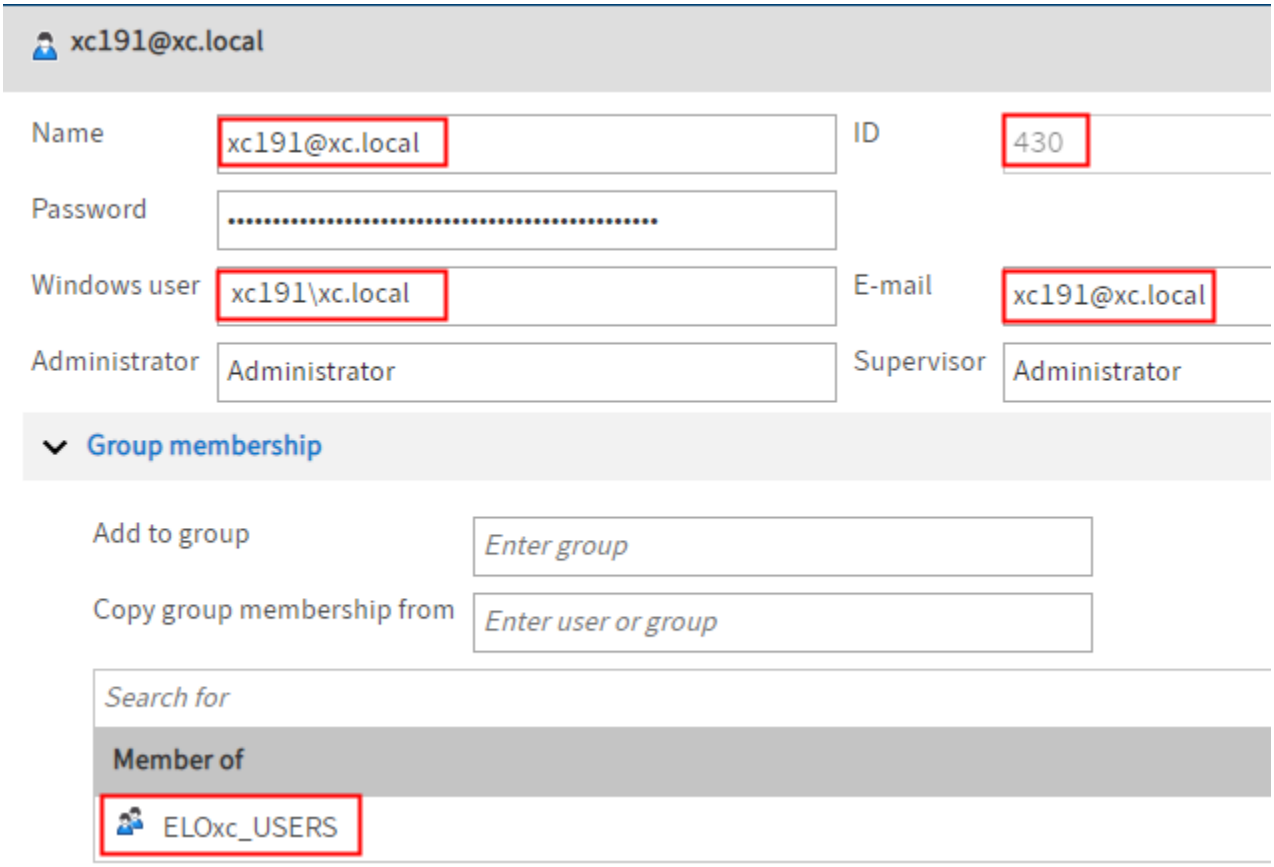
For example, if an alphabetical folder structure is required for filing e-mails from different mailboxes, you should create it in advance. If ELO XC creates the folder structure when

storing e-mails with user import, the access permissions are set according to what was filed first.

When you check whether the users have been imported in the ELO Administration Console, you will find both the user *xc191* and the user group *ELOxc_USERS*:



The user name *xc191@xc.local* is taken from the directory property *UserPrincipalName*. If this identity is selected, the *Windows user* is automatically set in the format *<domain>\<account>*. The SMTP address is applied as the e-mail address and the newly created user is automatically assigned to the *ELOxc_USERS* group stored in the configuration with the *group prefix* and *group membership* parameters:



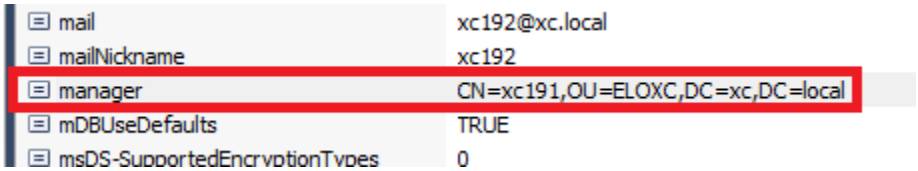
Information

The user ID cannot be changed because it is determined automatically by the ELO Access Manager. It is an internal identifier and plays an important role in user import by integrating

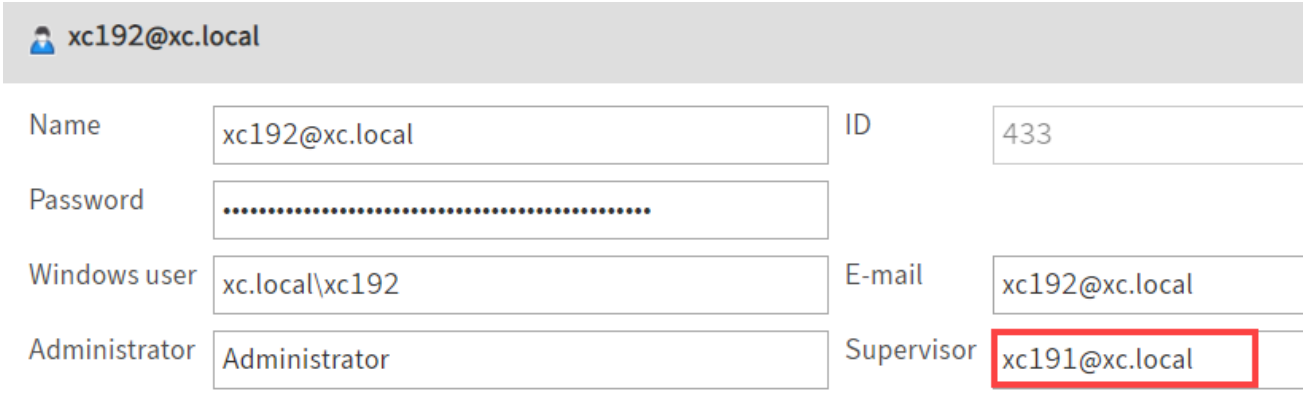
user data sets and permissions settings. It is the main reason why users previously imported from an external directory have to be identified in ELO. If this is not the case, permissions settings will be lost, orphaned, or incorrectly assigned. If users are imported with the ELO Indexserver LDAP connection or imported by ELO XC, the directory property *objectGUID* identifies the users. When importing users with LDAP in the ELO Administration Console, you need to make an alteration if you want to use the *objectGUID*.

When using LDAP mailbox catalogs, supervisors defined in ELO are automatically assigned to ELO users. This requires the LDAP property *manager*.

The user *xc192* is assigned *xc191* as supervisor in the local directory:



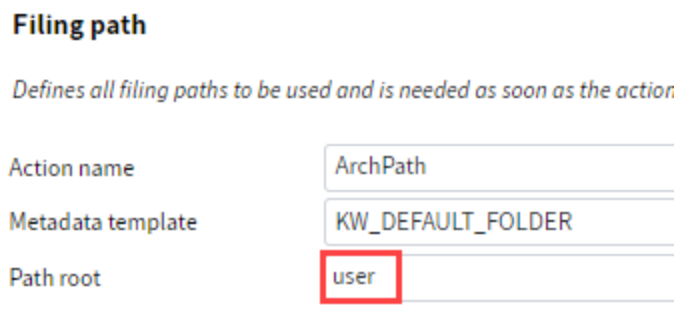
If user import is enabled, the ELO user *xc191* is automatically entered as the supervisor of ELO user *xc192* when this user is created:



If the *manager* property is not set, the *ELO Administrator* is entered as the supervisor. During subsequent processing, if the property is available or has changed, the setting will be applied.

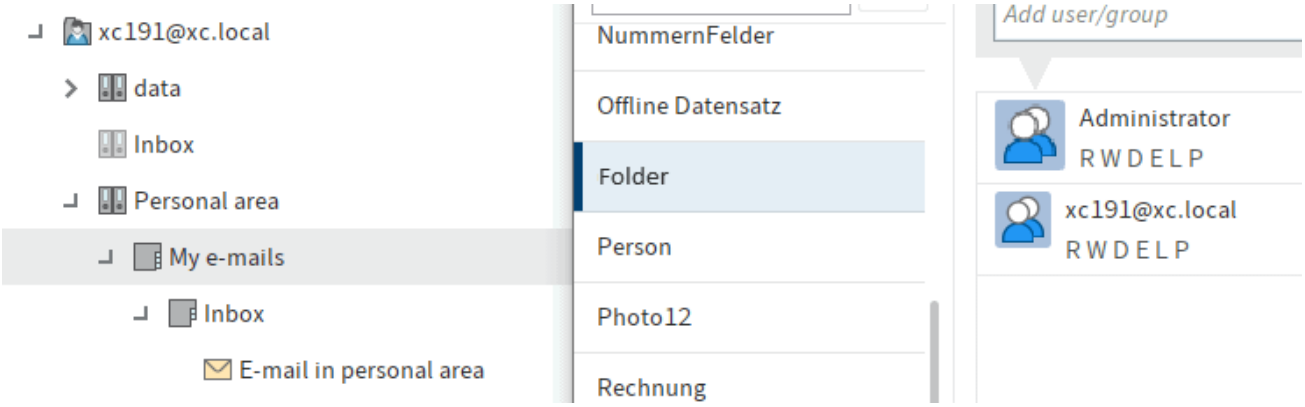
ELO user entries with automatic import

In order to apply the automatic user import settings to the user entries and to store e-mails in these structures, you need to modify the configuration of the filing path. In this case, you need to set *user* instead of the default value *archive*.



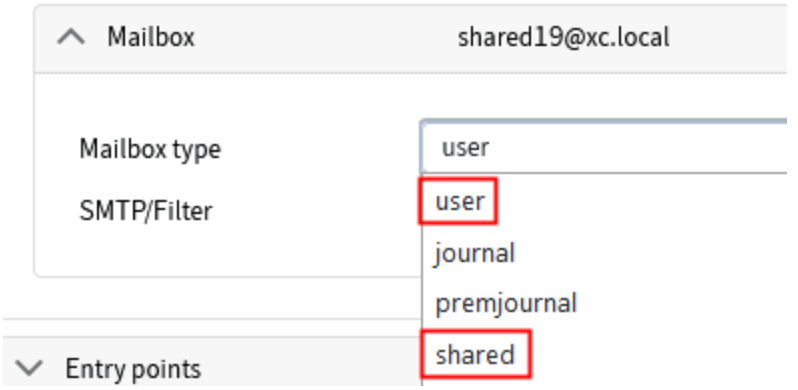
ELO user entries are part of the default configuration of a repository and are shared with all ELO users. Each user has a personal area in the repository, which cannot be accessed by other users. If you set the root of the filing path to *user*, the e-mails will be stored in this area. This eliminates the need to create a folder that is associated with a mailbox, which is why the filing path is easier to configure:

If you now store a message, you can see that the permissions settings of the user's personal area are automatically applied:

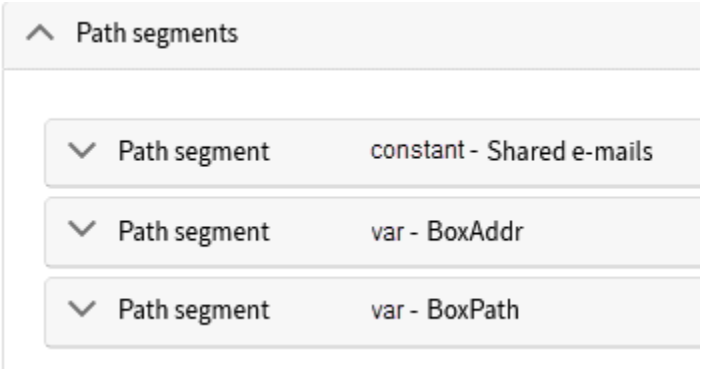


Shared mailboxes with automatic import

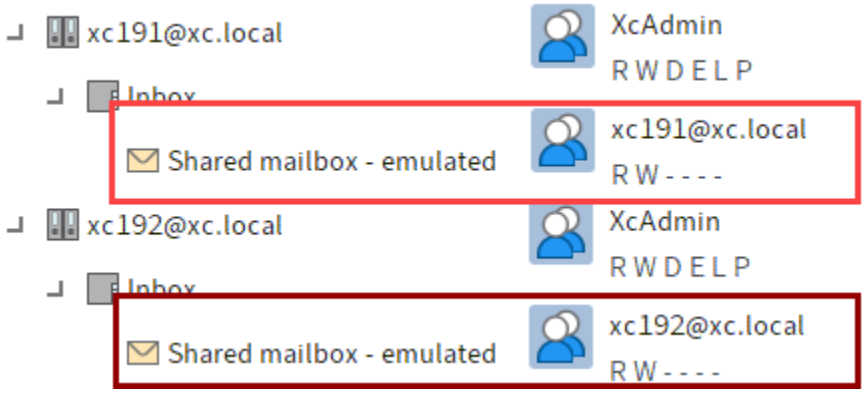
If you want to store messages from a *shared mailbox*, two different mailbox types are available – *shared* and *user*:



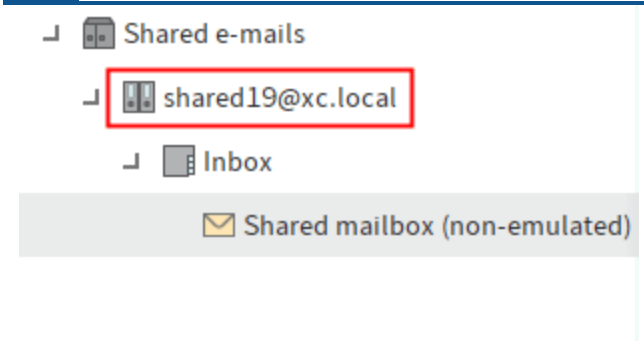
A shared mailbox *shared19* with configured delegates *xc191* and *xc192* is processed with the setting *shared*. This triggers mailbox emulation, which processes *shared19* as the mailboxes *xc191* and *xc192*. For this to work, the filing path must use a variable that is associated with the mailbox, indirectly associating the delegate as its owner. The path configuration could look like this:



If the e-mail is stored using the mailbox type *shared*, the result in the repository looks like this:



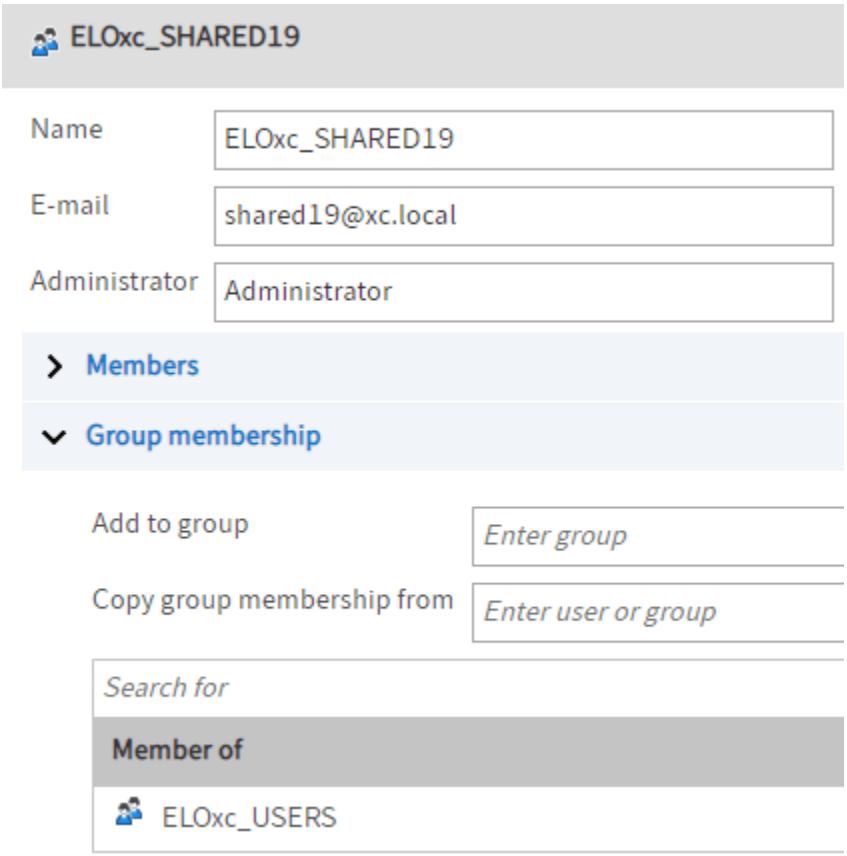
Each delegate receives a copy of the processed message. The permissions correspond to the configuration of the instance configuration. However, if the e-mail is stored with the mailbox *type="user"* in *native mode*, ELO XC uses this filing structure:



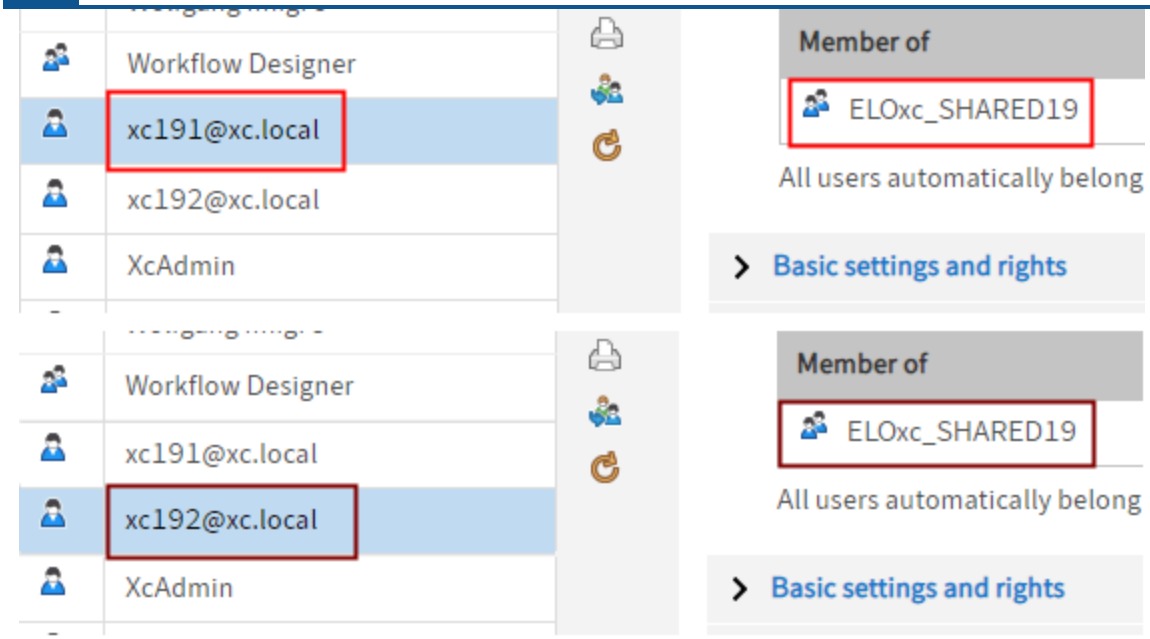
Only one message copy is stored for the shared and in this case not emulated, i.e. resolved mailbox *shared19@xc.local*.

In terms of the permission settings when processing a shared mailbox, regardless of whether you select the type *user* or *shared*, an ELO user group is always created whose members are the delegates of *shared19@xc.local*. The difference is that *type="shared"* assigns individual access rights and *type="user"* assigns group rights.

In the ELO user manager, you can see that a user group called *shared19@xc.local* was created for *shared19@xc.local*. This user group is a member of the general group *ELOxc_USERS*. This occurs independently of *type="shared"* and *type="user"*, reflecting the mailbox configuration and probably the mailbox usage.

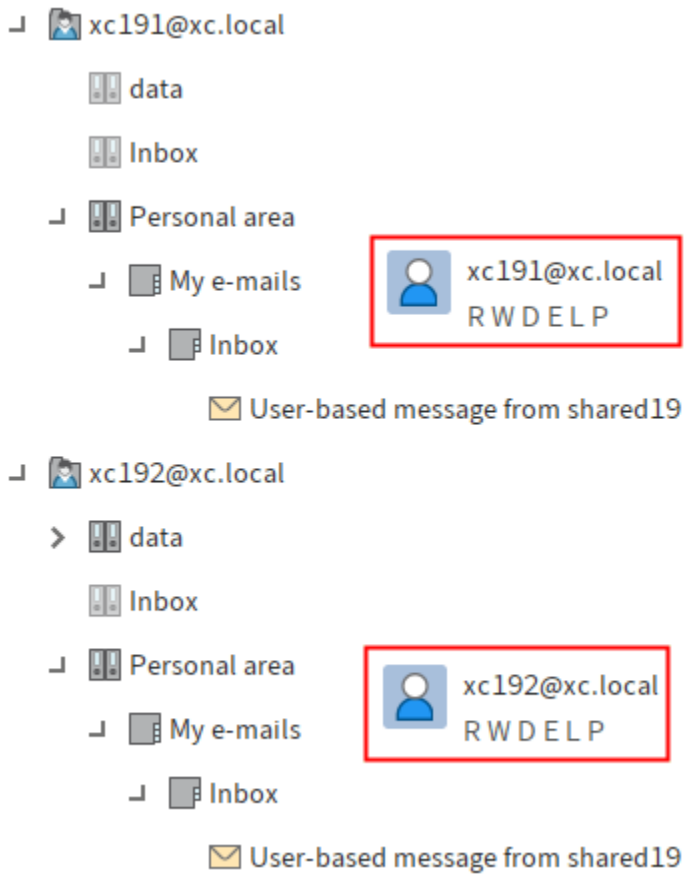


The mailbox delegates are assigned to the user group *shared19@xc.local*:

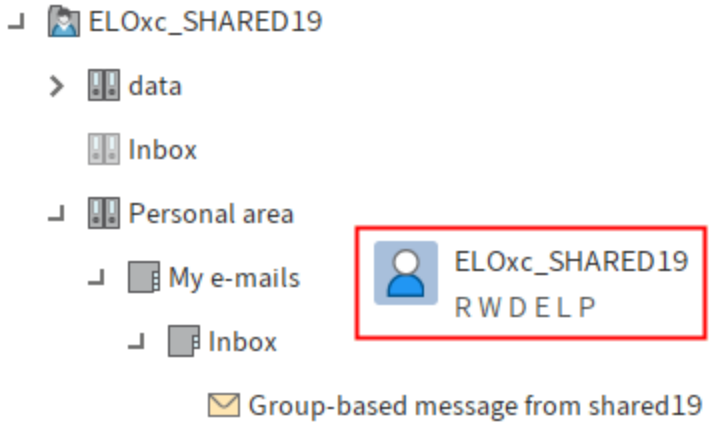


If you use mailbox emulation in native mode, the result for the path root *user* is congruent to the result for the path root *archive*.

This is what the result with mailbox emulation looks like:



This is the result with native mode, i.e. without mailbox emulation:

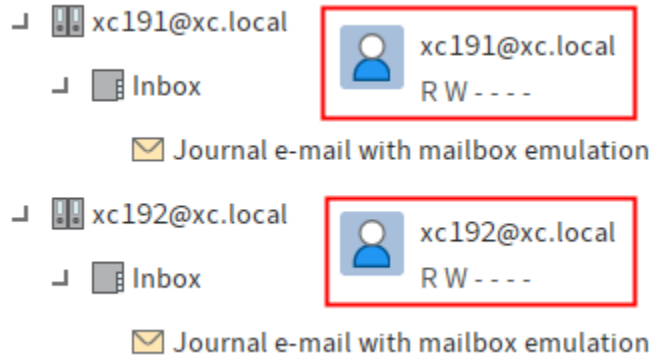


Journal recipients with automatic import

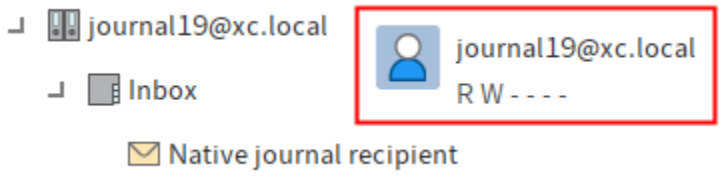
User accounts are also automatically transferred when processing journal recipients. However, a journal is not used in the same sense as shared mailboxes. A journal recipient is a user mailbox and is therefore treated by ELO as a user.

As long as a journal is configured as the mailbox type *journal* in ELO XC, mailbox emulation with automatic creation of all users whose mailboxes are on the same Exchange server mailbox database will occur as expected. However, if the journal is processed in native mode, user accounts are created in ELO for all journal mailbox owners as well as the user account of the journal recipient.

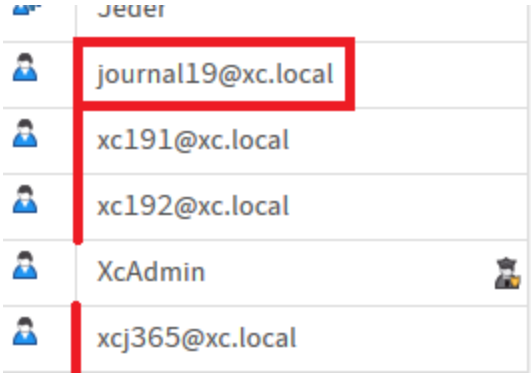
E-mail storage with mailbox emulation (*type="journal"*) of the journal recipient *journal19@xc.local*:



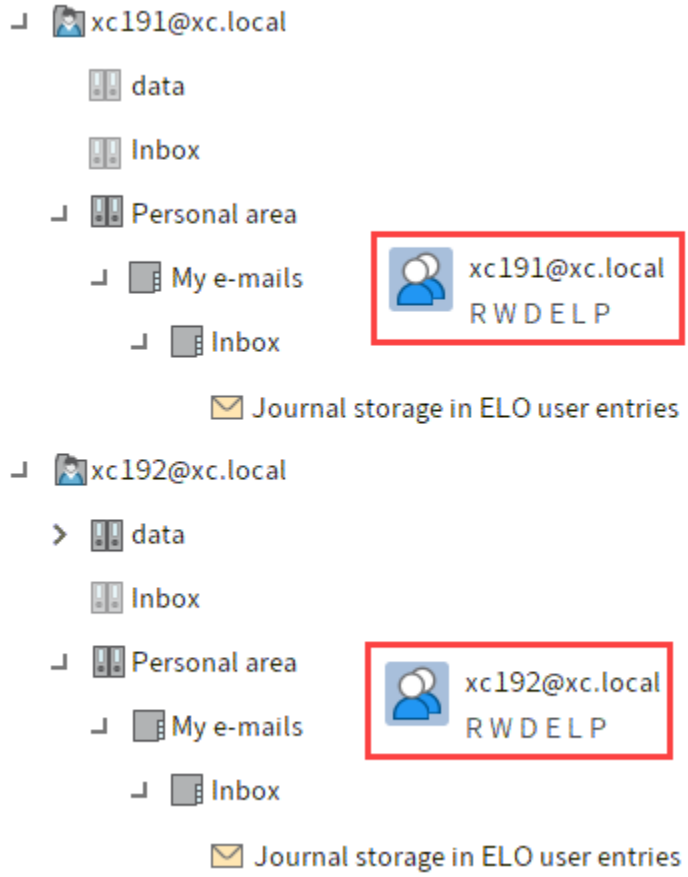
E-mail storage with *type="user"* in native mode produces this result:



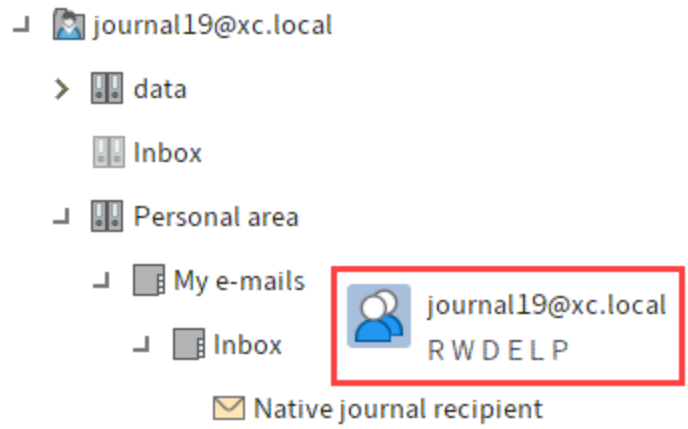
Due to native mailbox processing, the journal recipient and the journal mailbox owners appear in the user manager:



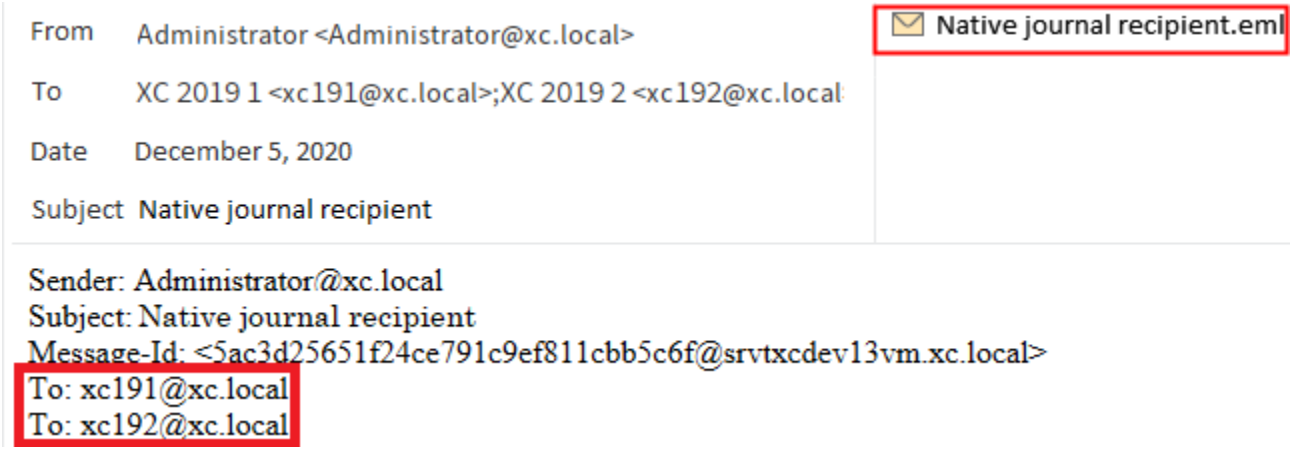
If an e-mail is stored in the ELO repository using mailbox emulation and the path root *user*, it is stored in the user's personal area:



With native processing (*type="user"*) of the journal recipient in ELO, only one message copy is stored in the repository for *journal19@xc.local*.



E-mail storage of journal recipients in native mode differs from native mode with shared mailboxes not only because the user *journal19* is also transferred. In this case, other messages are stored as well:



The original message is stored in the journal recipient's mailbox as an attachment to an envelope. The envelope contains the recipients of the message. E-mail storage of journal recipients in native mode is therefore more suited for meeting requirements for documentation of e-mail correspondence than is user-based e-mail storage. If you still want ELO users to be able to access these messages at a later date, it is possible to extract the original message using downstream processes and make them available to automatically imported ELO users with appropriate permissions settings.

Mailbox catalogs with automatic import

The different catalog types in ELO XC all retrieve the main user properties *Name*, *Windows user*, and *E-mail*. ELO XC also applies the GUID (*objectGUID* property) of the external mailbox owner as the ELO user GUID. If users are created manually, there is no external GUID so you need to configure the LDAP import function in the ELO Administration Console to use this GUID. The external GUID is essential for data transfer later on, which is why it is also specified as the default setting for the ELO Indexserver LDAP connection.

In addition to the main user properties, ELO XC stores additional catalog data in the *ldapuserprops* table of the ELO Access Manager, which can be retrieved via the Indexserver API.

The table below illustrates how properties are mapped according to the access constants in the Indexserver API:

> **Information**
>
> In the *ps* column, the external names correspond to PowerShell naming conventions.

External name	ldap	m365	UserInfoC
displayName	X	X	LDAP*KEY*DISPLAY_NAME
distinguishedName	X	X	LDAP*KEY*DISTINGUISHED_NAME
legacyExchangeDN	X	X	LDAP*KEY*LEGACY*EXCHANGE*DN
Mail	X		LDAP*KEY*MAIL
PrimarySmtpAddress		X	LDAP*KEY*MAIL
Manager	X		LDAP*KEY*MANAGER
msExchRecipientTypeDetails	X		LDAP*KEY*EXCH*RECIPIENT*TYPE_DETAILS
RecipientTypeDetails		X	LDAP*KEY*EXCH*RECIPIENT*TYPE_DETAILS
	person	person	LDAP*KEY*OBJECT_CLASS
objectGUID	X		LDAP*KEY*OBJECT_GUID
Guid		X	LDAP*KEY*OBJECT_GUID
	false		LDAP*KEY*ONLINE
		true	LDAP*KEY*ONLINE
sAMAccountName	X		LDAP*KEY*SAM*ACCOUNT*NAME
samAccountName		X	LDAP*KEY*SAM*ACCOUNT*NAME
userPrincipalName	X	X	LDAP*KEY*USER*PRINCIPAL*NAME

It is possible to configure external properties as main user properties (*Name*, *Windows user* and *E-mail* using the *Identity format* parameter, although *Name=UserPrincipalName*, *Windows user=domain\account*, and *E-mail=SMTP address* should remain the preferred default.

Name=account and *Windows user=UserPrincipalName* can be useful if you have older repositories. User data is only created if it is missing and is not updated after that, so you must carefully consider whether ELO Access Manager databases should do without the preferred default settings.

ELO Indexserver LDAP connection

With the default settings, the ELO Indexserver LDAP connection adopts the following properties:

LDAP	UserInfoC
Cn	LDAP*KEY*CN
displayName	LDAP*KEY*DISPLAY_NAME

LDAP	UserInfoC
distinguishedName	LDAP*KEY*DISTINGUISHED_NAME
	LDAP*KEY*DOMAIN
groupType	LDAP*KEY*GROUP_TYPE
mailNickname	LDAP*KEY*MAIL*NICK*NAME
manager	LDAP*KEY*MANAGER
msExchMailboxGuid	LDAP*KEY*MS*EXCHANGE*MAILBOX_GUID
name	LDAP*KEY*NAME
objectCategory	LDAP*KEY*OBJECT_CATEGORY
objectGUID	LDAP*KEY*OBJECT_GUID
objectSid	LDAP*KEY*OBJECT_SID
proxyAddresses	LDAP*KEY*PROXY_ADDRESSES
sAMAccountName	LDAP*KEY*SAM*ACCOUNT*NAME
sAMAccountType	LDAP*KEY*SAM*ACCOUNT*TYPE
userAccountControl	LDAP*KEY*USER*ACCOUNT*CONTROL
userPrincipalName	LDAP*KEY*USER*PRINCIPAL*NAME

Information

Refer to the LDAP connection documentation for instructions on how to customize the property mappings. Nevertheless, it makes sense for the LDAP connection and the user import from ELO XC to generate the same result for the main ELO user properties *Name*, *Windows user*, and *E-mail*.

Summary

ELO XC can only set the permissions of newly created items in the repository. Once data is archived, the settings in the repository cannot be changed. Access permissions to items in the repository must always be set with the client. Automatically transferred users are retained even if they are subsequently deleted in the directory. This also applies for the LDAP property *manager*.

With the default configuration, it is necessary to define the settings, including the permission settings, for storing e-mails in the repository. The default metadata forms *Folder* and *E-mail* are useful configuration tools and are available as standard metadata templates of ELO XC.

Automatic import of user accounts is an option that allows you to avoid manual configuration of filing structures. The mailbox owners are imported as ELO users and assigned access rights that correspond to the mailbox type depending on the configured processing type. The main user properties *GUID*, *Name*, *SMTP address* and *Account* or *UserPrincipalName* can be applied for all catalog groups. Furthermore, additional directory data is stored in ELO Access Manager.

When mapping external data to user properties in ELO, however, make sure that you avoid conflicts since, for example, manual creation of users with simultaneous transfer by ELO XC can cause ambiguities that you may not notice until later on and are difficult to correct.

It is therefore important that you answer the following questions when creating or rebuilding a repository:

- How will users be managed in the long term? Where will they be managed?
- Will there only be ELO users, or will all user data be imported from an external source?
- Have you made sure that ELO Administrator and ELO Service (and other service accounts, if applicable) are not generated from external sources? If the main user accounts are to be managed externally, have you made sure that their passwords remain changed so as not to disable active ELO components?
- Which component meets the administrative requirements for automatic user transfer: LDAP connection, LDAP import, or ELO XC?

The key aspects of permissions configuration for e-mail storage as illustrated in this example are:

- ELO Administration Console: Creation and validation of users and groups
- LDAP connection: Import of user accounts (general and at granular level)
- Metadata forms and fields: Default metadata forms *Folder* and *E-mail*
- Metadata templates: *KW*DEFAULT*FOLDER* and *KW*DEFAULT*DOCUMENT*
- Mailbox types: *user*, *shared*, *journal*
- Import of user accounts: Activation, identity format, assume ownership
- Filing paths: Path root *archive* or *user*, variables associated with a mailbox

The mailbox type *user* is always associated with native processing. The mailbox types *journal* and *shared* can also be processed in native mode, but we do not recommend this because the user focus is lost as a result. The recommended default for these two mailbox types is mailbox emulation.

The results/implications of automatic user import are listed in the following table:

Mailbox	Configuration	Path root	Permissions settings
User	*user*	*archive*	User-based
		user	User-based
Shared	*shared*	*archive*	User-based (plus own group)
		user	User-based (plus own group)
	user	*archive*	Group-based via own group
		user	Group-based via own group
Journal	*journal*	*archive*	User-based
		user	User-based
	user	*archive*	Mapped to journal recipient
		user	Mapped to journal recipient

Aspect form for e-mails

Introduction

Aspect forms were first released as a preview with ELO 21.01. They will replace the standard metadata forms in ELO in the long term. Aspects were introduced as an advanced way to classify data by group name. Aspects are field groups and the associated aggregates, whereas the index groups associated with standard metadata forms only permit assignment of single values. This example contains instructions on how to configure an aspect form for ELO XC as an alternative to the standard *E-mail* metadata form.

Aspect

In ELO, an aspect can be interpreted as a user data aspect (i.e. documents and folders). For example, invoices could be defined by an address aspect, but also by an an e-mail aspect (if the invoice was received by e-mail). The address is likely to contain a first name, last name, street address, postal code, and town, while an e-mail contains the names and SMTP addresses of the recipients. User data can therefore be interpreted and evaluated according to different criteria, which can be organized and implemented using aspects. Technically speaking, an aspect is a named field set.

Aspect field

An aspect can contain multiple fields of a type. For an *address*, the street or house number would be single aspect fields. An aspect contains at least one field. The available field set is determined by all defined aspect fields.

Aspect mappings

An aspect mapping establishes a link between a form and aspects and determines their cardinality, i.e. whether these aspects are used individually or as a list.

Example: The e-mail address is an independent aspect determined by the *Display name* and *SMTP address* field set. This aspect can be used for the sender and the three possible recipient lists *To*, *Cc*, and *Bcc*.

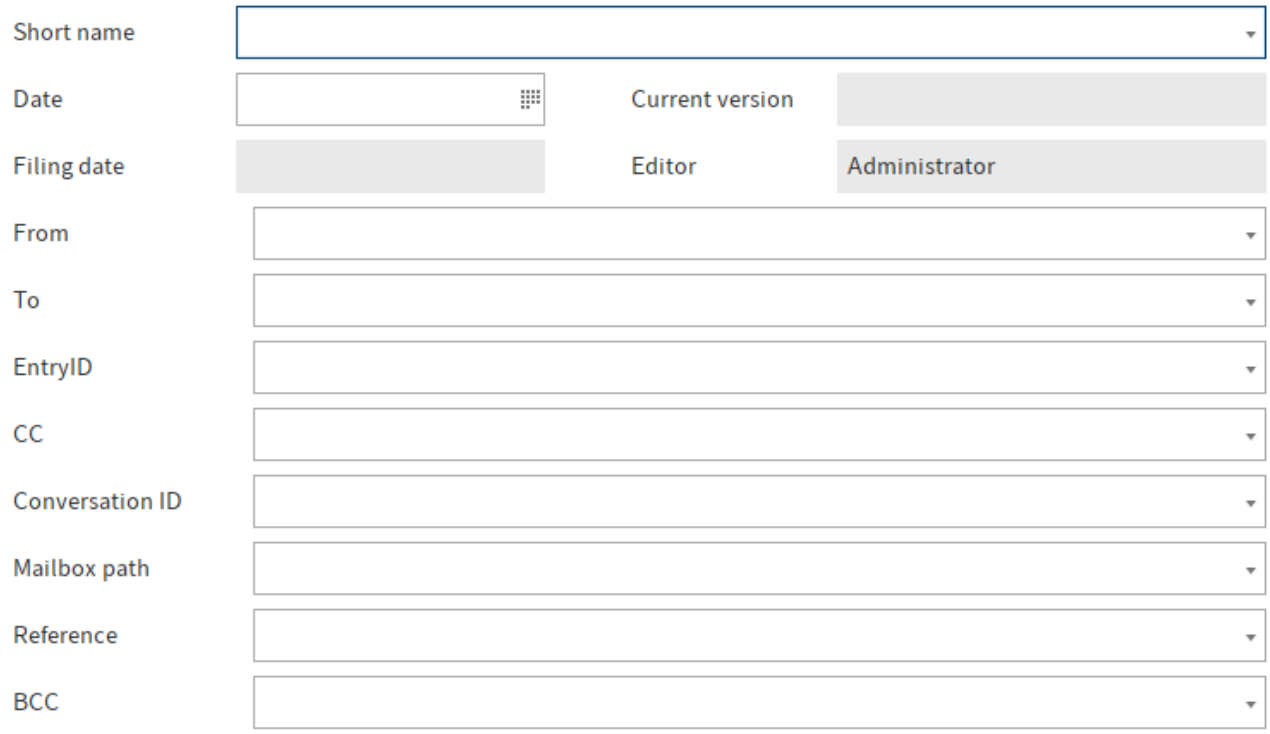
From	Max	max@dot.com
	Hugo	hugo@gmail.com
Cc	John	j.smith@outlook.astro
	Group Central	gc.group@domain.de

If you want to evaluate an e-mail based on the recipient aspect, it is sufficient to organize the data according to the associated aspect. You would store each recipient irrespective of their role, since an e-mail address is always an e-mail address, whether it is used to send or receive e-mails.

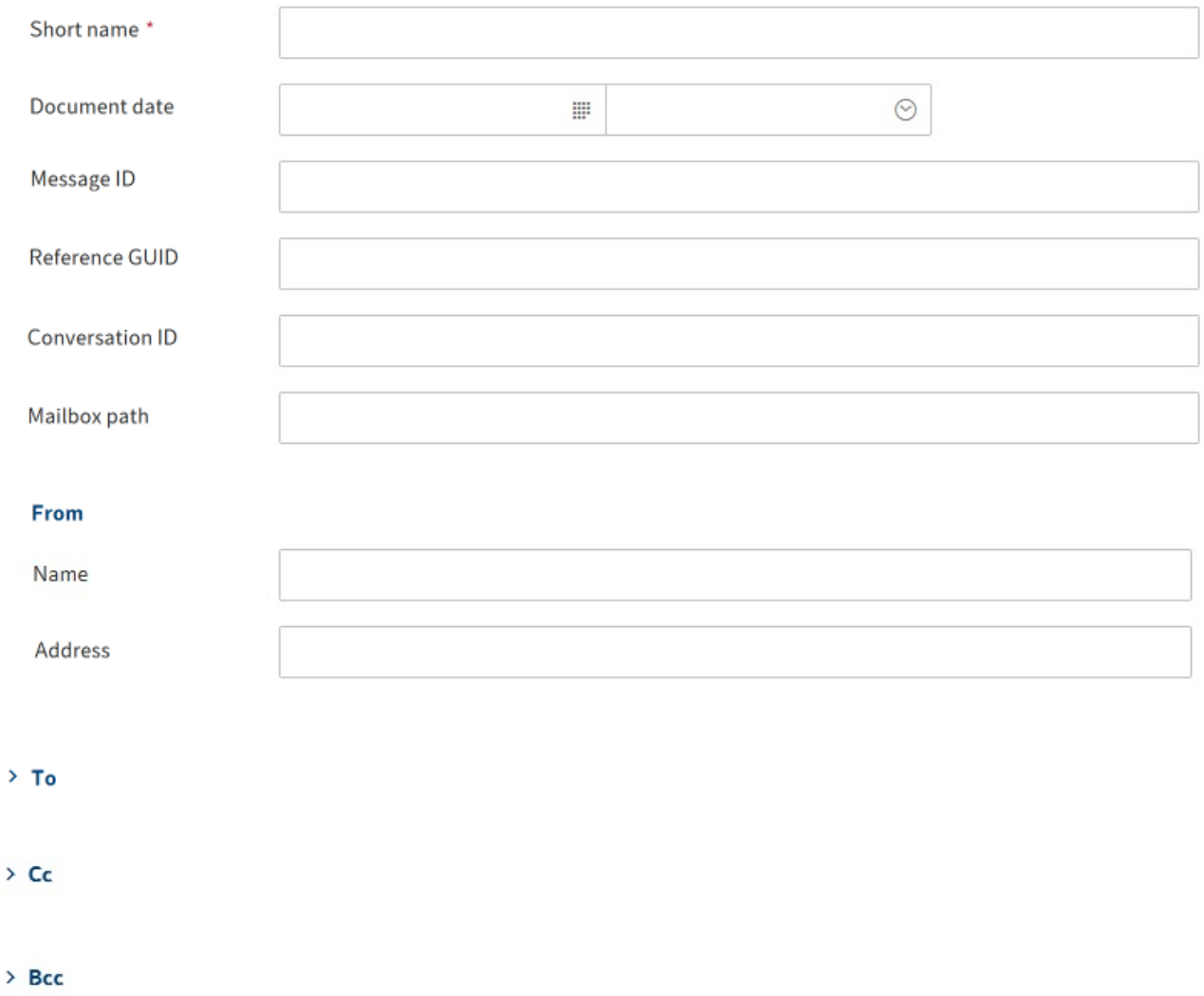
The role needs to be configured as an aspect mapping in the form. The *From* mapping would use the aspect individually, and the *To*, *Cc*, and *Bcc* mappings would use the aspect in a list as an aggregate.

The following configuration steps show how to build a corresponding aspect form according to the structure of the standard e-mail metadata form.

Source: 'E-mail' metadata form



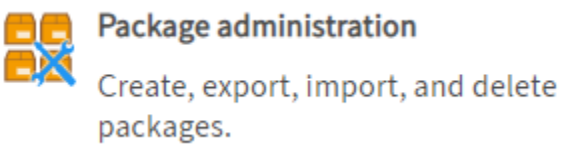
Target: 'E-mail demo' aspect form



Create aspect form

Support for aspect forms

Aspect forms are part of packages and are managed with the ELO Administration Console.



You can create a new package in the *Package administration* menu.

Package

XC_PACK

Delete package

Namespace *

DEMO

Name *

XC_PACK

Description

Alternative standard e-mail metadata form demo

Maintainer

Icon

Upload

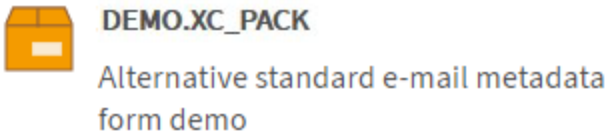
GUID

(819EE13A-5CC8-04A8-6589-48C51443191F)

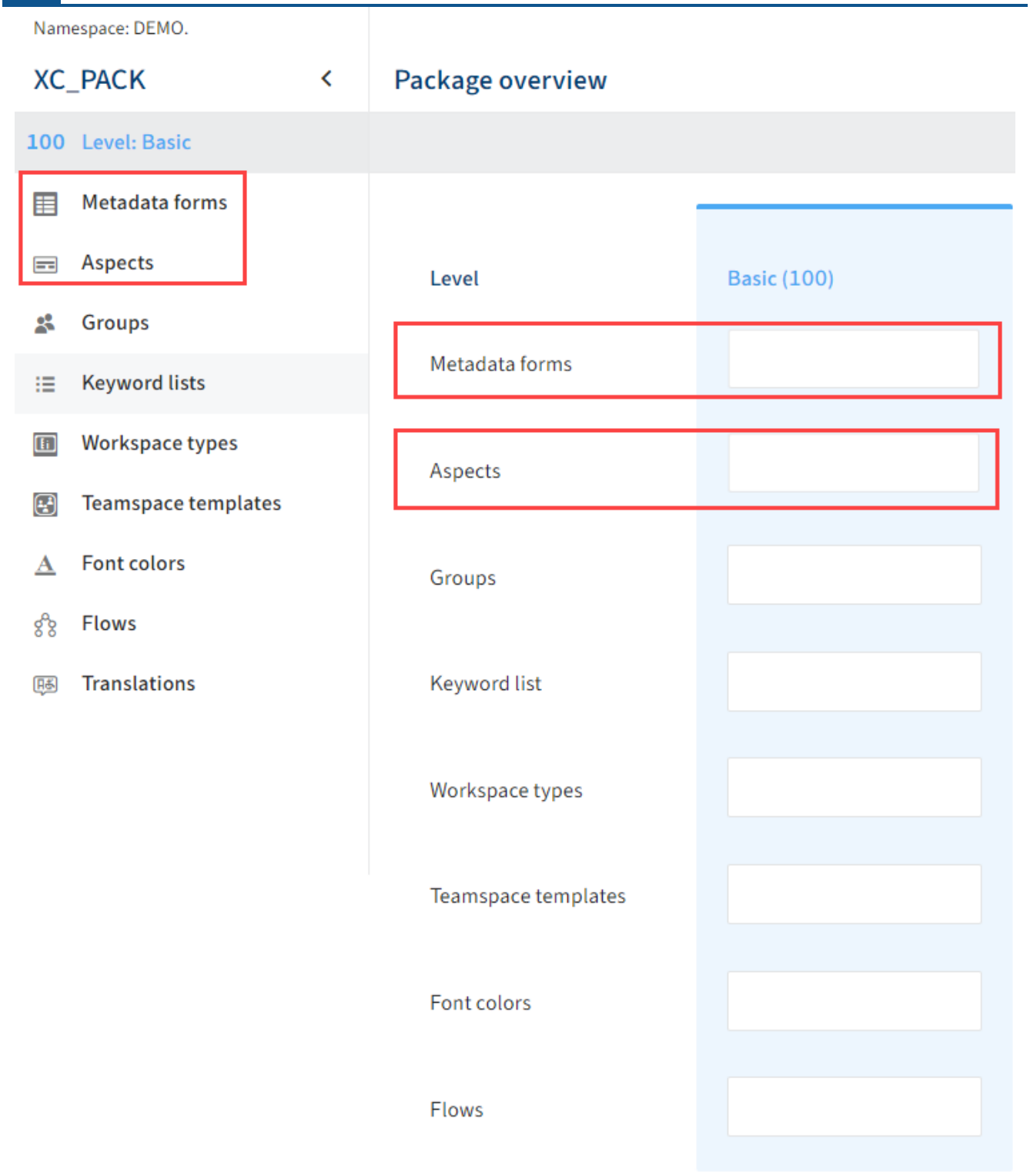
There are unsaved changes.

Save Cancel

Afterwards, the package will appear on the home screen of the ELO Administration Console. You can now configure it.



You need to configure aspect forms under the *Metadata forms* and *Aspects* entries:

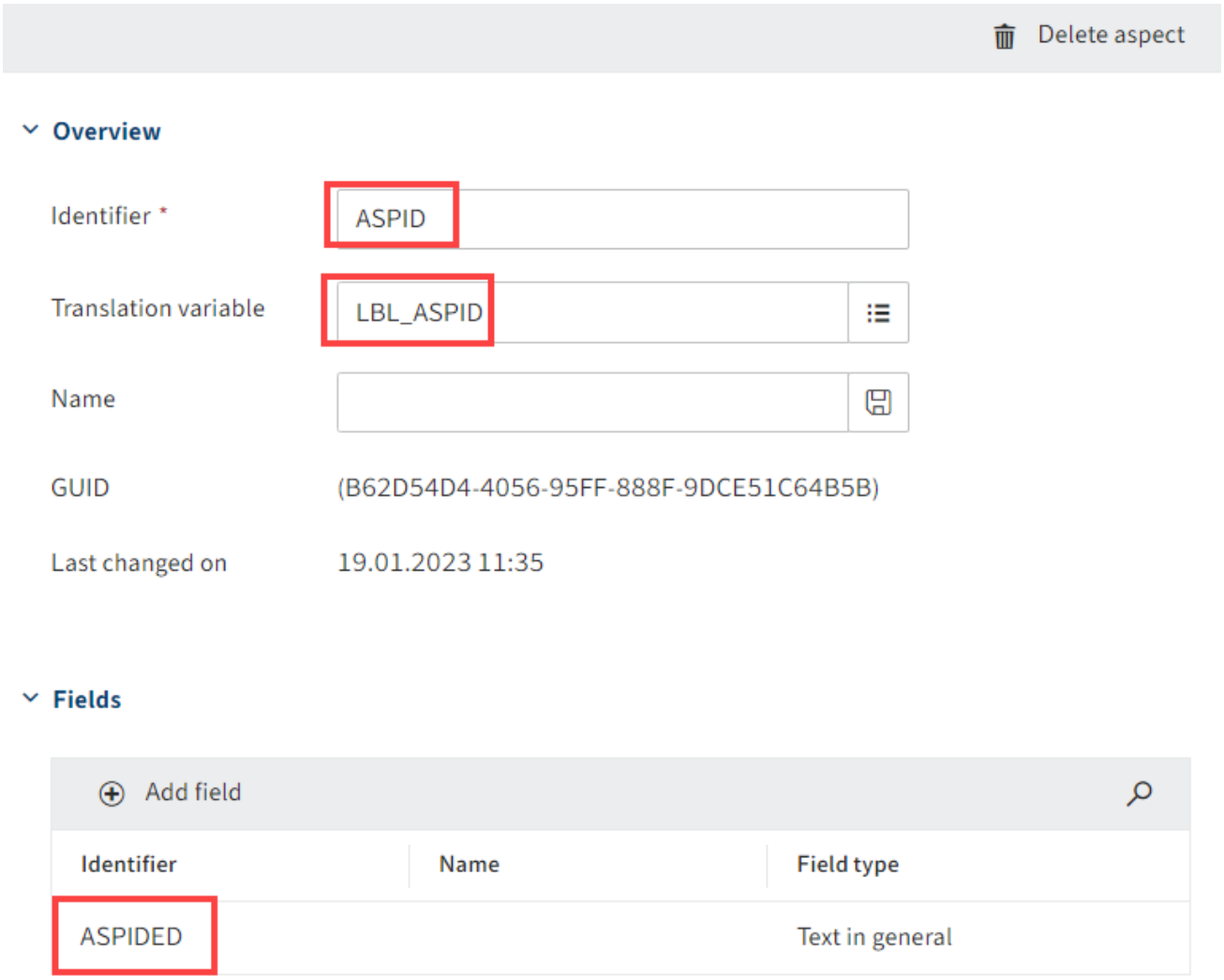


Aspects

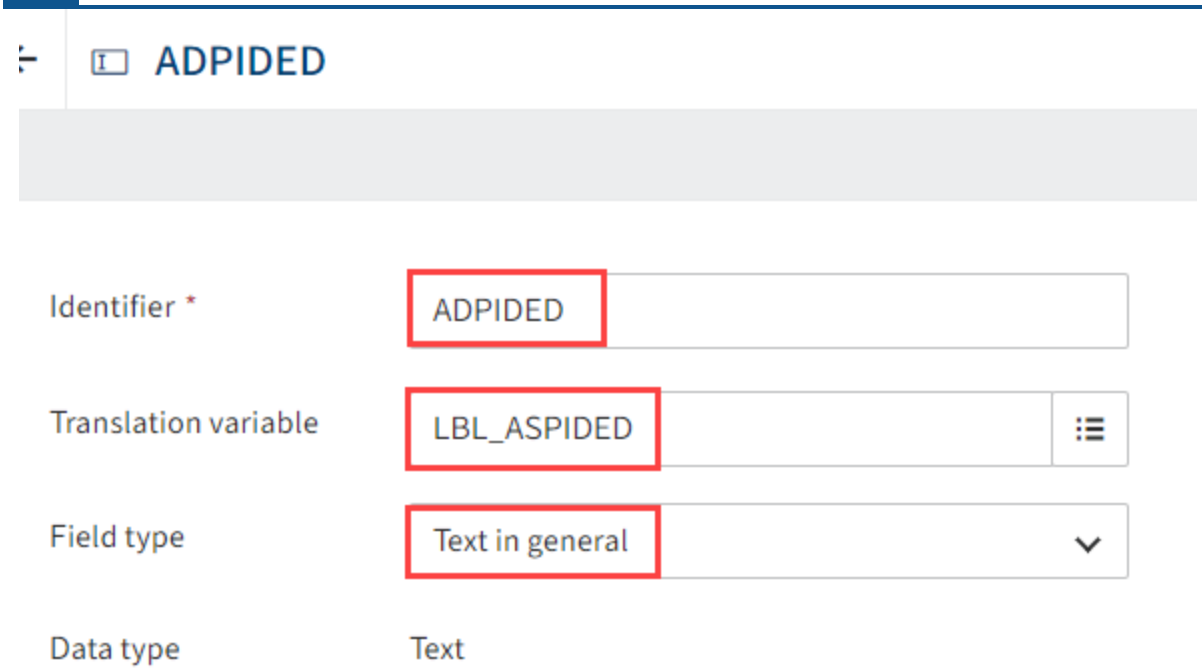
First, you need to configure the aspects. This involves defining aspects and the associated fields, which can then be assigned to forms.

Under *Aspects*, you can add new aspects.

Example using an e-mail ID:

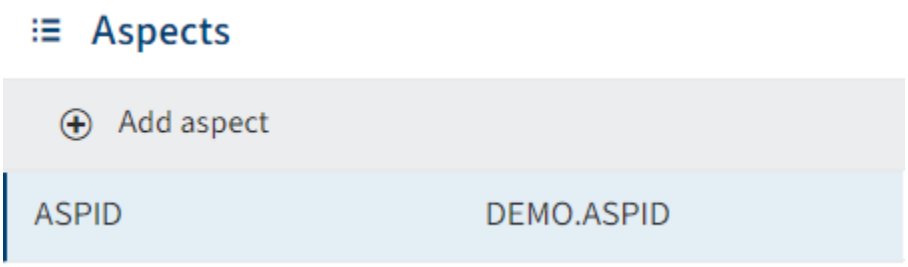


The aspect is assigned an identifier, a text association for localization purposes, and a data field. Configure the data field as follows:

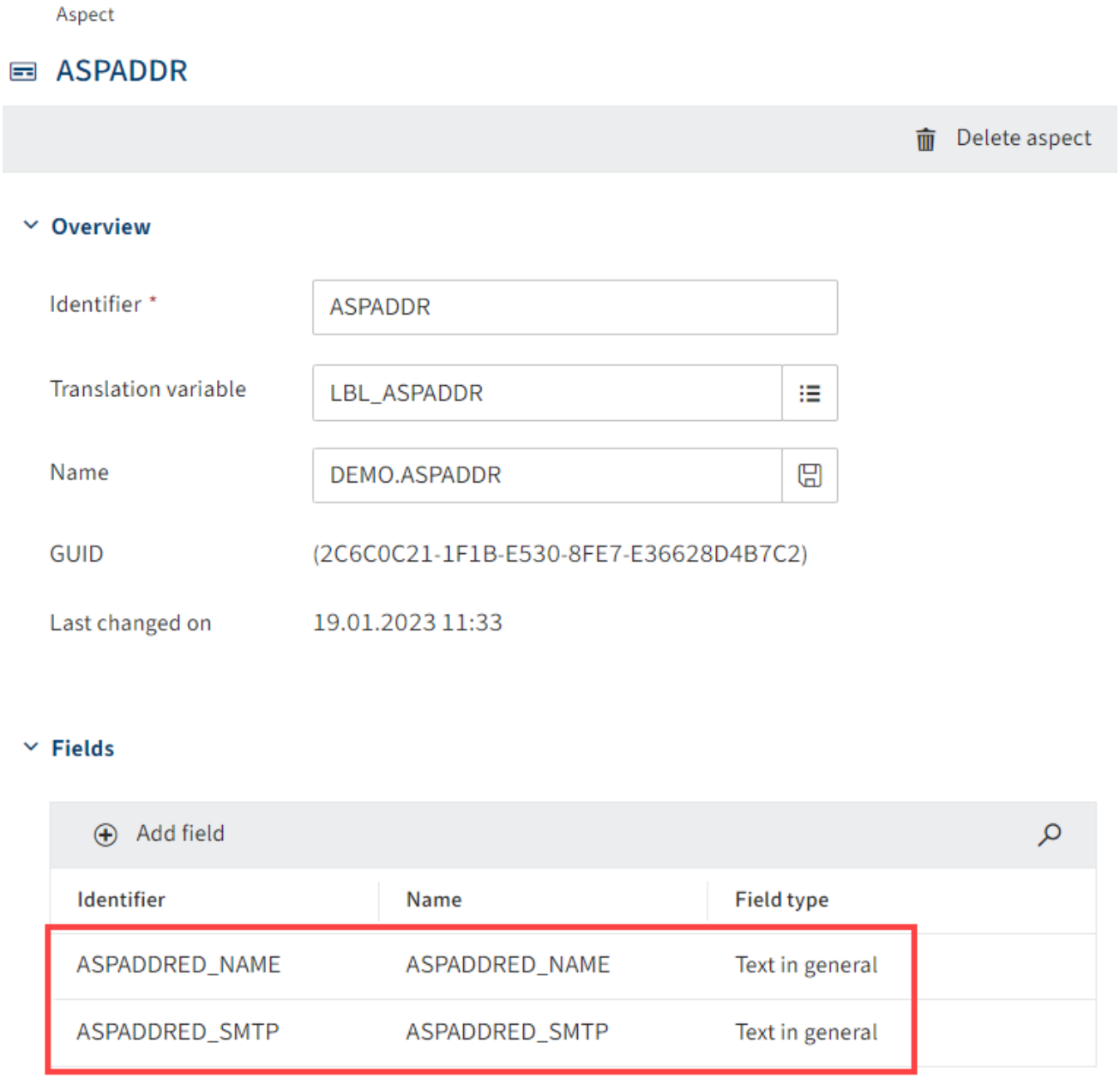


This example only uses text fields. Aspects are identified by the prefix *ASP* and the runtime localization is identified by the prefix *LBL*. It is possible to use an own name.

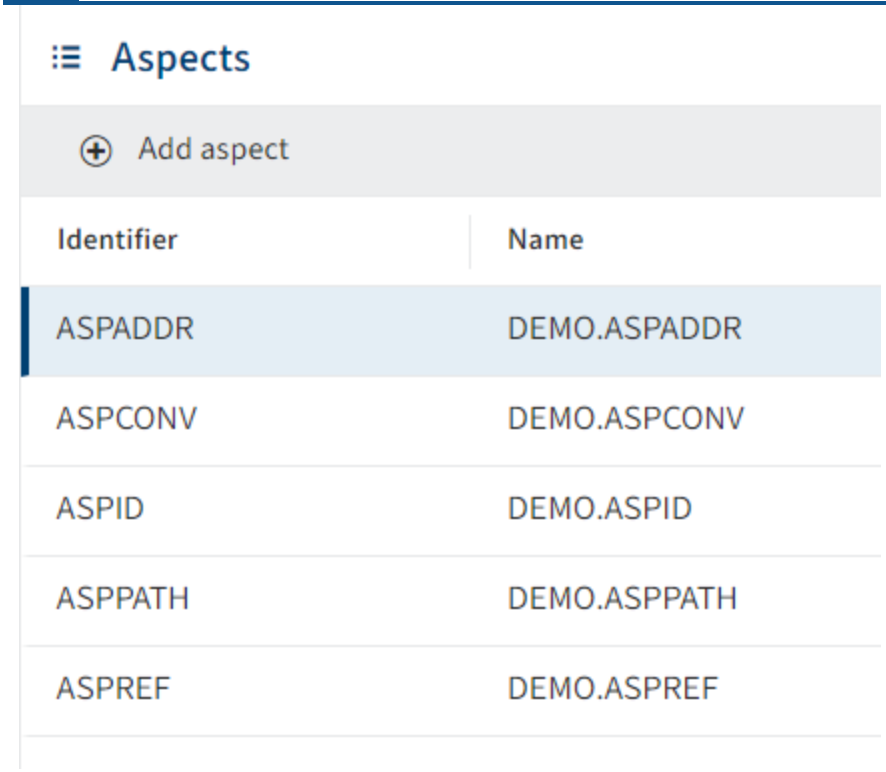
Finally, you need to save the settings of the new aspect. Afterwards, the aspect appears in the list.



The central aspect for e-mails is required for recipients, which can behave differently. A recipient can have a name as well as the SMTP address, and the recipients can assume different recipient roles in a single e-mail.

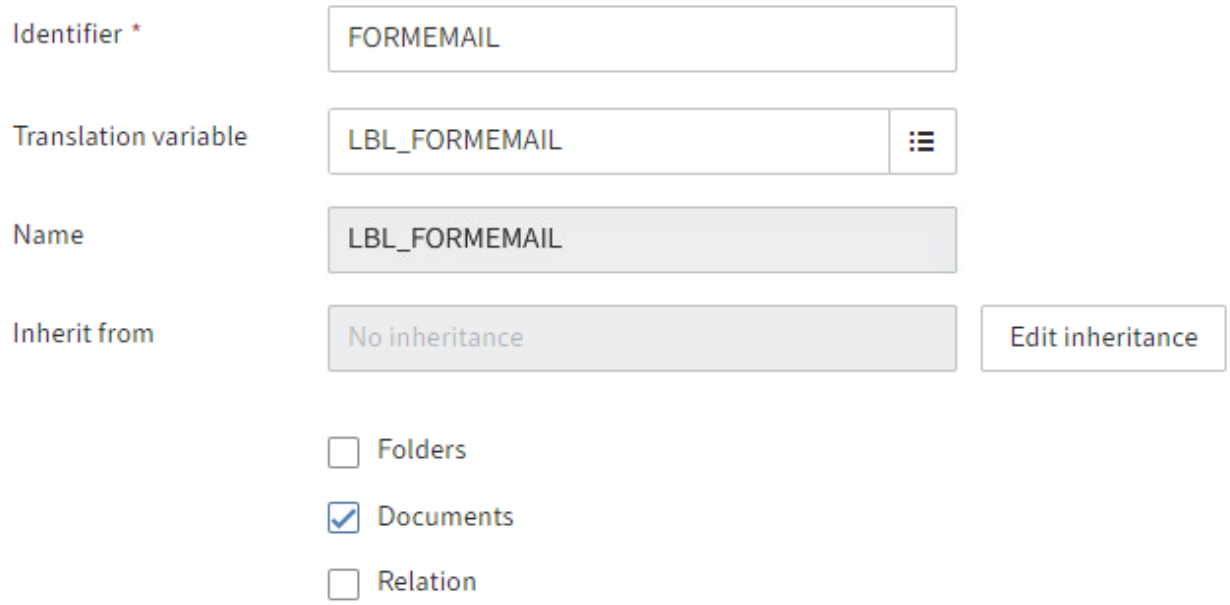


To configure the standard e-mail metadata form for use with aspects, we also need aspects for the reference GUID, the mailbox path, and the correspondence. These aspects must be available before we create the form with the required mappings.



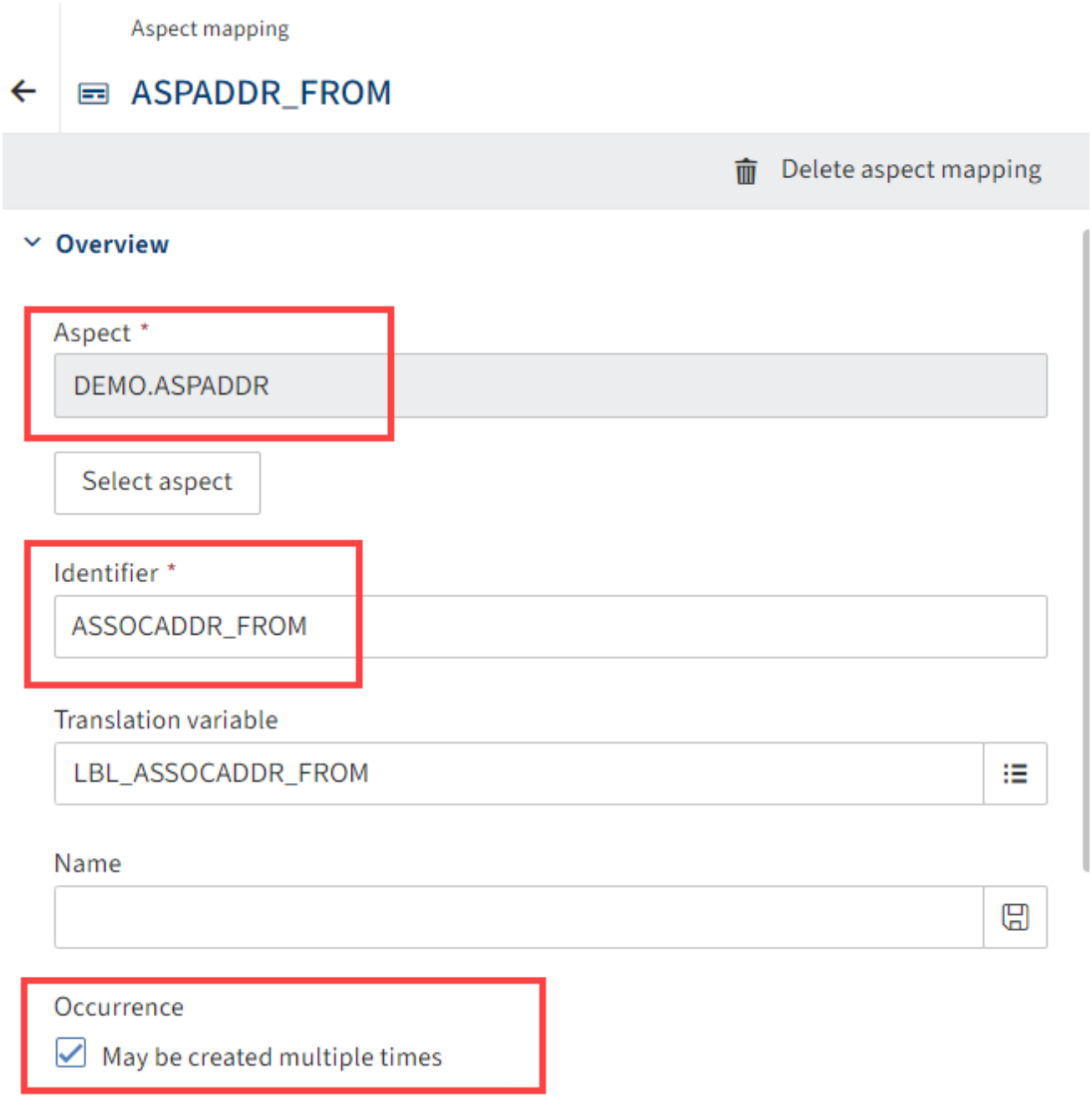
Metadata form

Now you can create the form with the required aspect mappings. First, create the form:

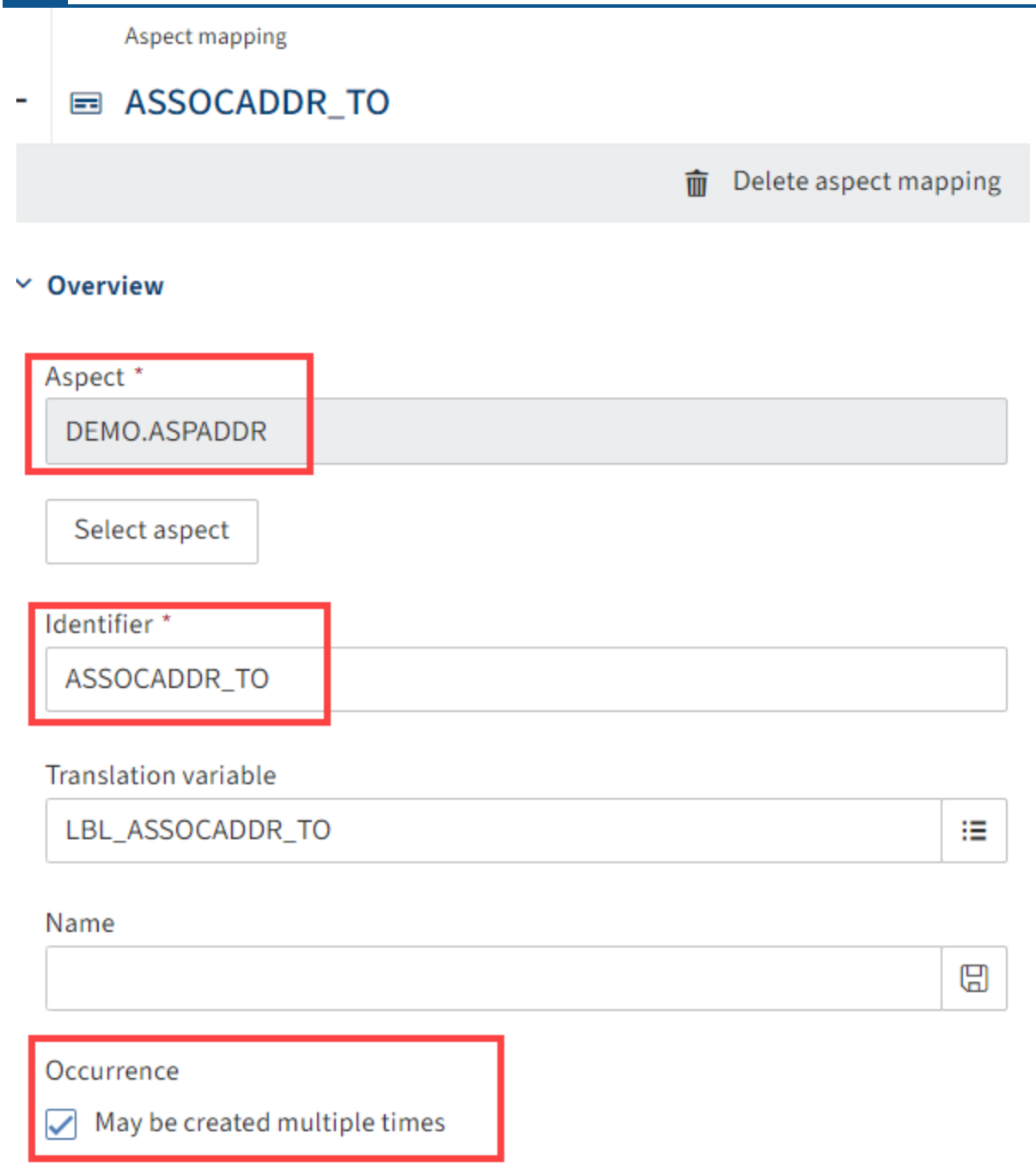


Afterwards, you need to assign the aspects to the form and specify the view type. Whereas all aspects are used individually in an e-mail, it is necessary to distinguish between an individual sender and the various recipient lists. This requires four mappings.

Sender:



Recipient list:



Once you have created the appropriate mappings for the other recipients and all other aspects, the form configuration should end up looking like this:

Aspect mappings

Add aspect mapping

Identifier	Name	Occurrence
ASSOCPATH	ASSOCPATH	OPTIONAL
ASSOCREF	ASSOCREF	OPTIONAL
ASSOCADDR_FROM	ASSOCADDR_FROM	OPTIONAL
ASSOCADDR_TO	ASSOCADDR_TO	OPTIONAL_MANY
ASSOCADDR_CC	ASSOCADDR_CC	OPTIONAL_MANY
ASSOCADDR_BCC	ASSOCADDR_BCC	OPTIONAL_MANY
ASSOCCONV	ASSOCCONV	OPTIONAL
ASSOCID	ASSOCID	OPTIONAL

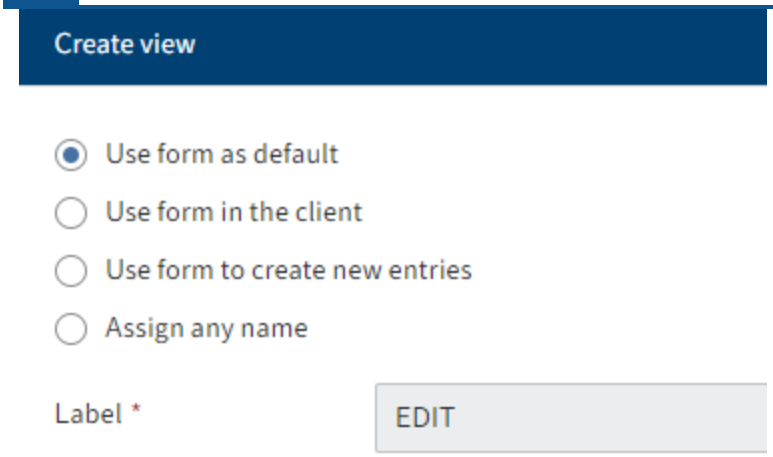
The view types of the mappings were selected so that the form can also be used if individual field values are not available (*optional*). The aspects (data structures) and the form mappings (usage type of data structures) are now configured. Before the form can be used, you have to create at least one view.

Views

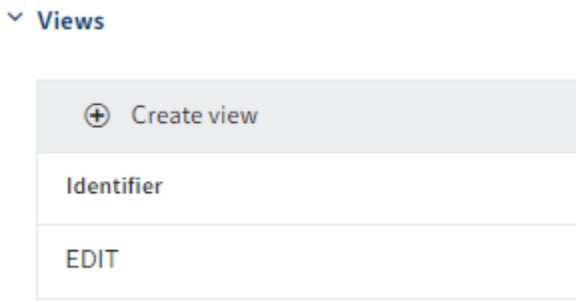
Create view

Identifier
No contents available

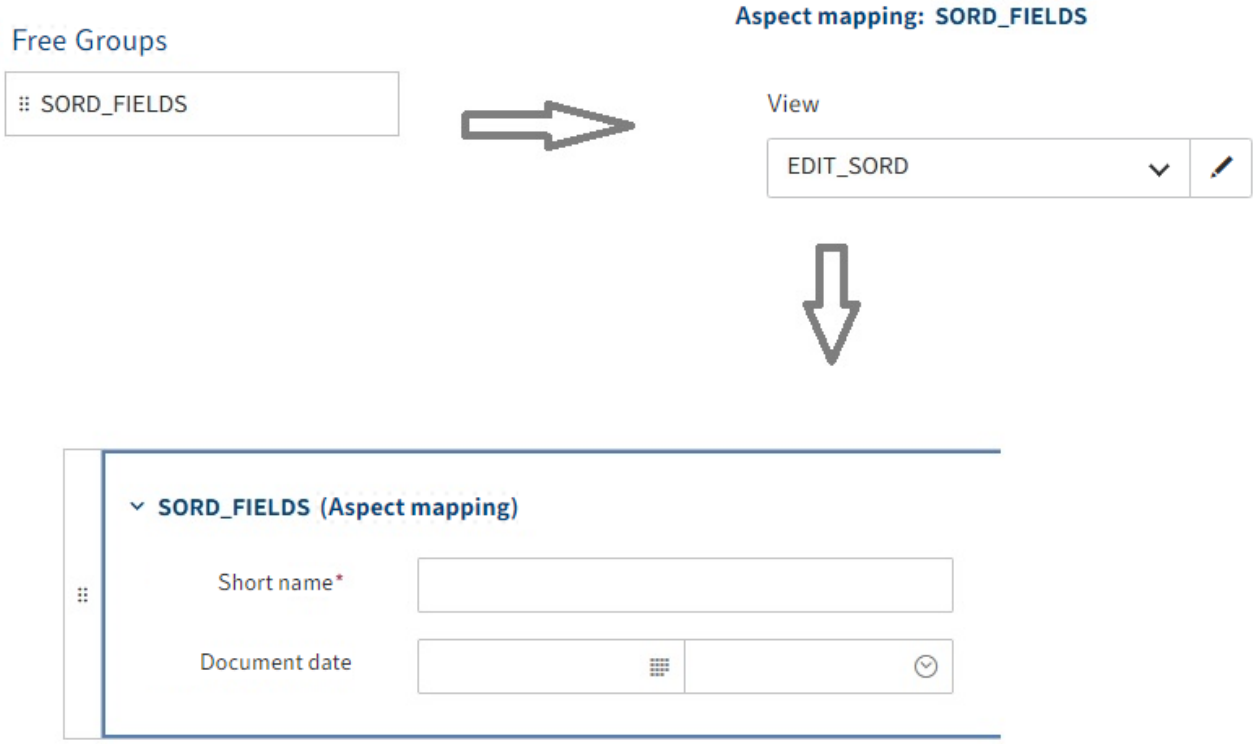
The form should be used as the default:



Afterwards, the view appears in the list of metadata form properties where it can be edited:



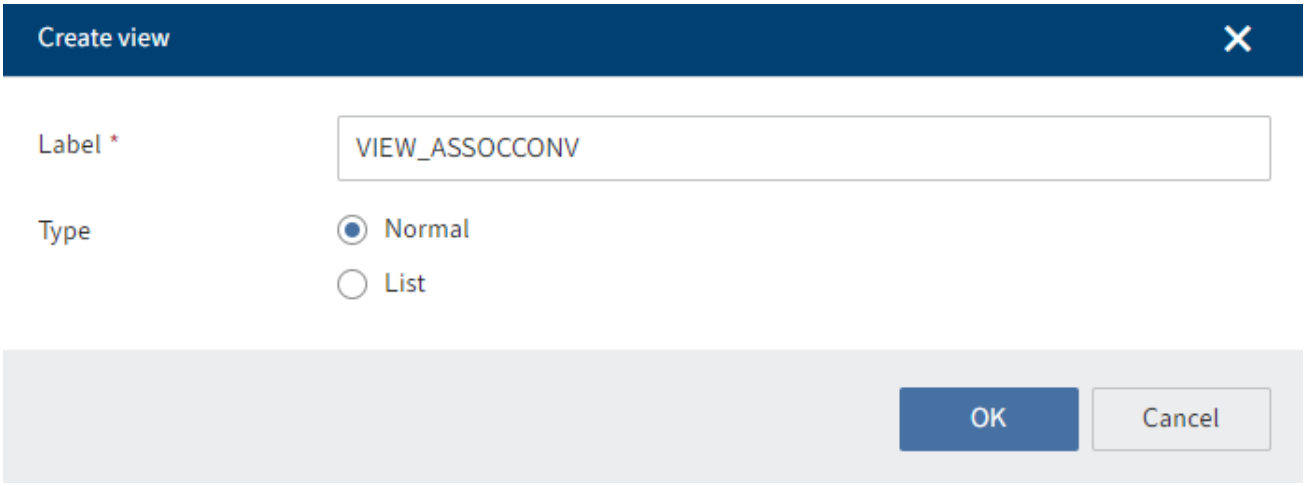
First, you need to add the SORD fields:



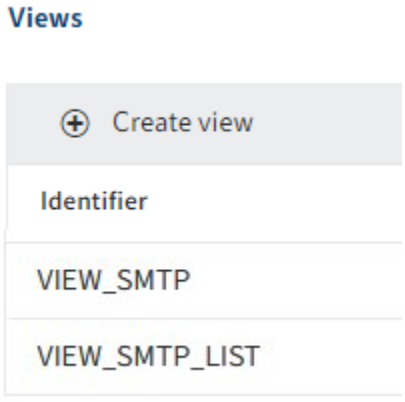
Afterwards, you can add the relevant mappings for e-mails one by one.

You can specify the view type while creating the fields.

Example:

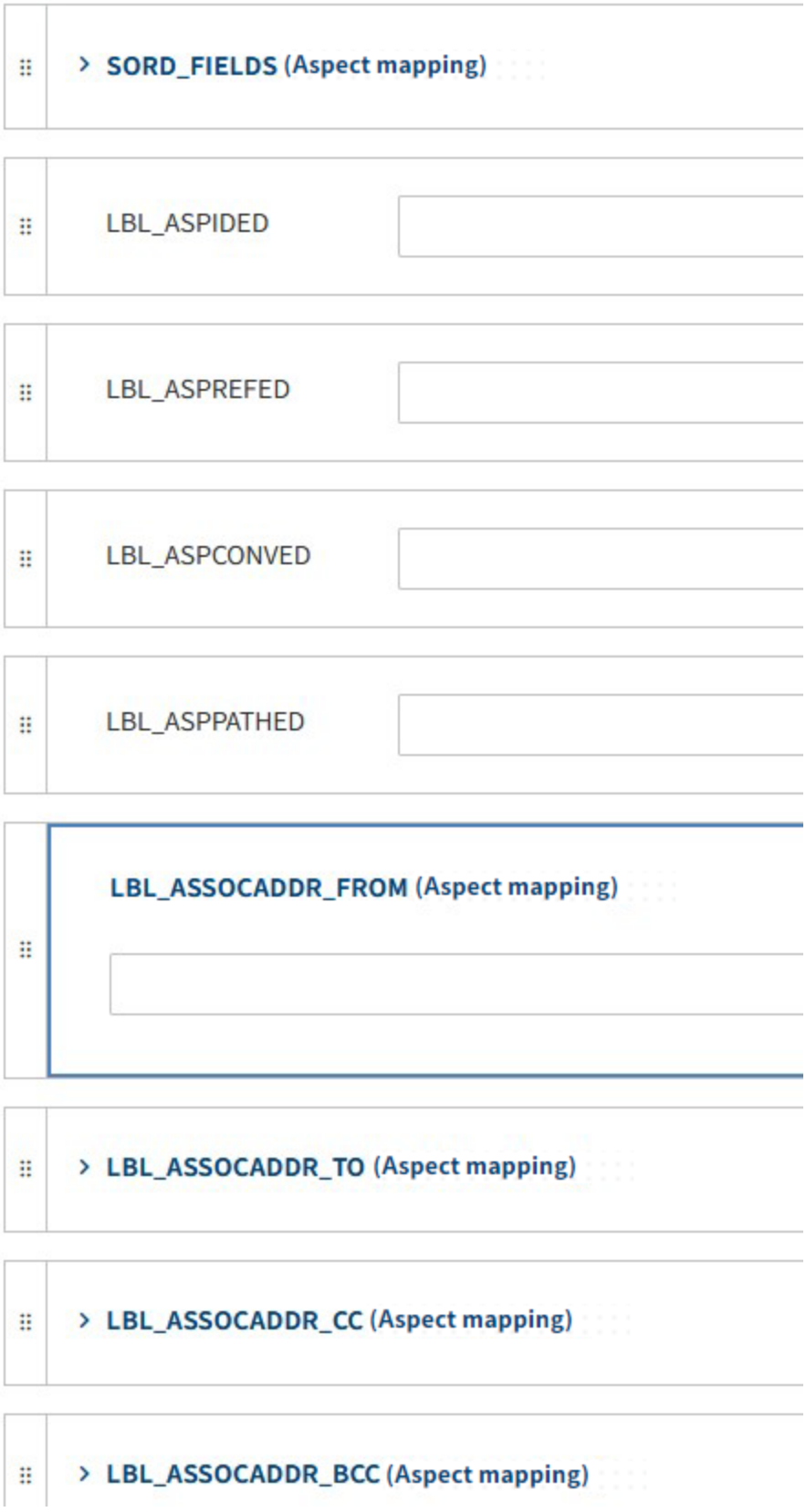


In the case of e-mails, you should select the type *Normal* for all mappings except *To*, *Cc*, and *Bcc*. These should be assigned the *List* type. Since *From* with the *Normal* type and *To*, *Cc* or *Bcc* use the same aspect, two different views are needed, which should be displayed automatically depending on the mapping. The views can be created in direct relation to the aspect so that they are available for the mapping. The views for the *ASPADDR* aspect look like this:



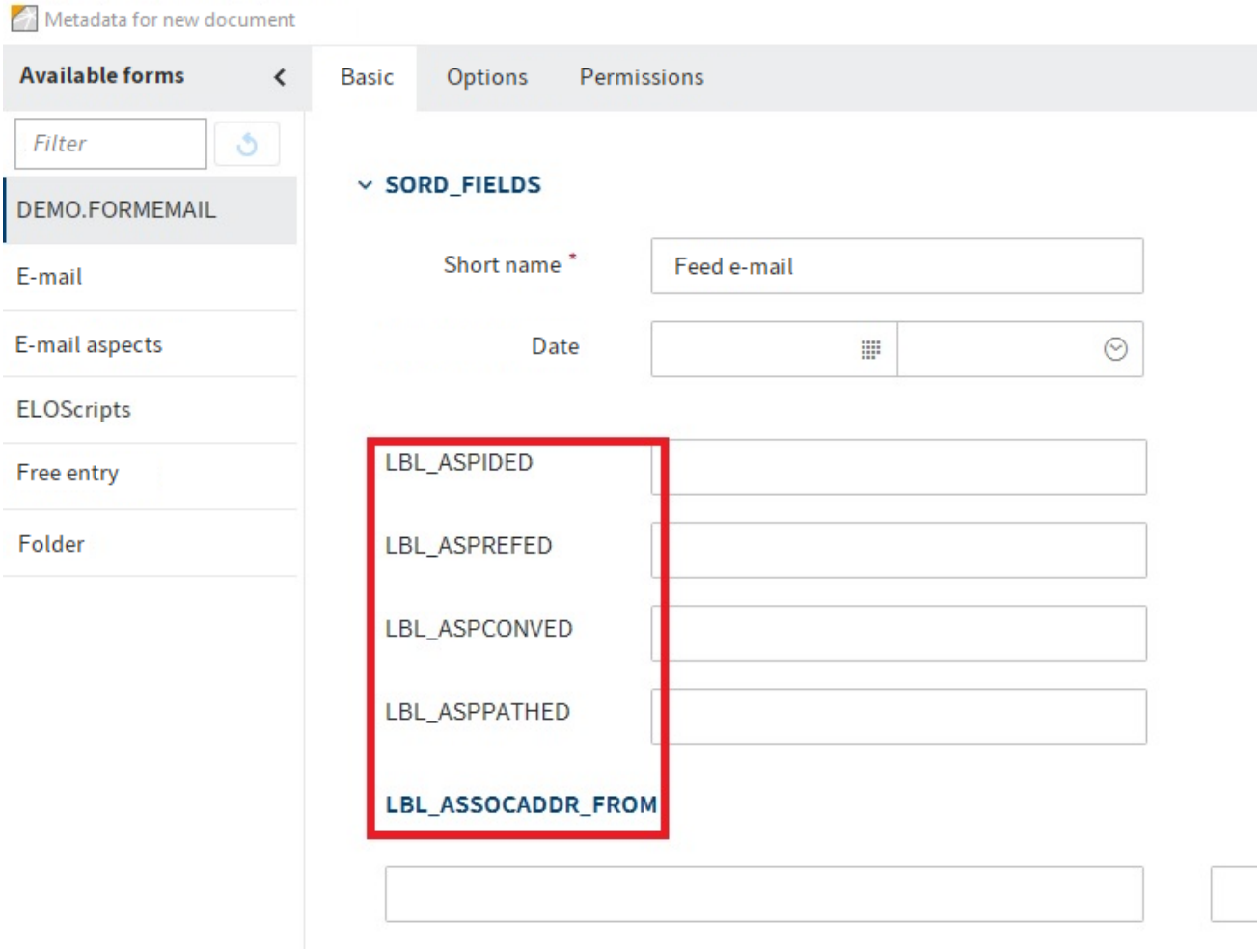
It makes sense to consider whether you want to use the display texts of the fields or the mapping. In the case of some fields, a custom view is the suitable option. The *ASPADDR* aspect is a pair of fields that is displayed separately and in lists, so it makes more sense to display the names of the mapping in addition to the field names.

Once you have completed the configuration, the new form should look like this:

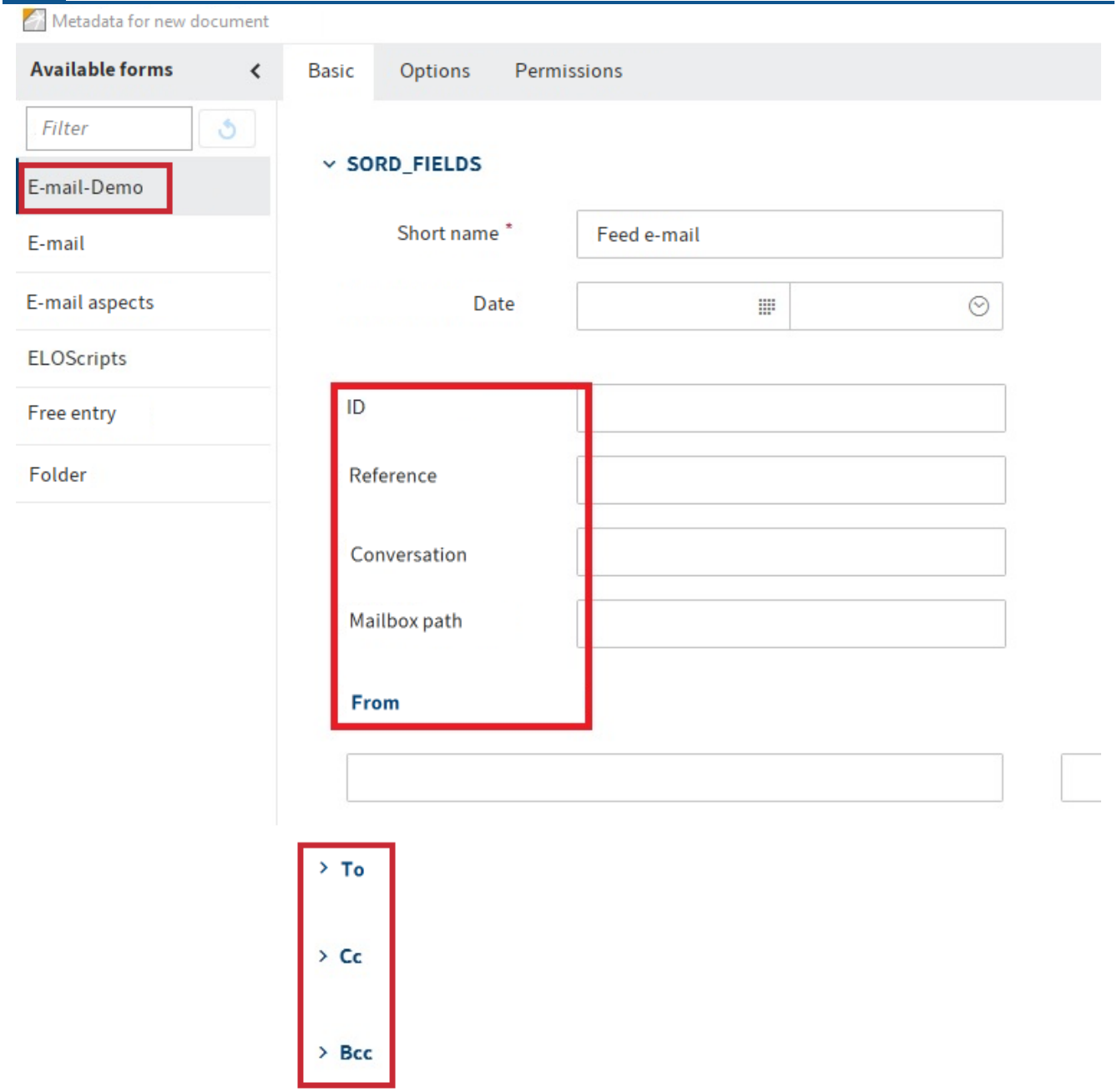

SORD_FIELDS (Aspect mapping)
LBL_ASPIDED
LBL_ASPREFED
LBL_ASPCONVED
LBL_ASPPATHED
LBL_ASSOCADDR_FROM (Aspect mapping)
LBL_ASSOCADDR_TO (Aspect mapping)
LBL_ASSOCADDR_CC (Aspect mapping)
LBL_ASSOCADDR_BCC (Aspect mapping)

Display

After saving the form, you can test the result in the ELO client. You can see that the display texts are not configured yet and that the technical name is still being used.

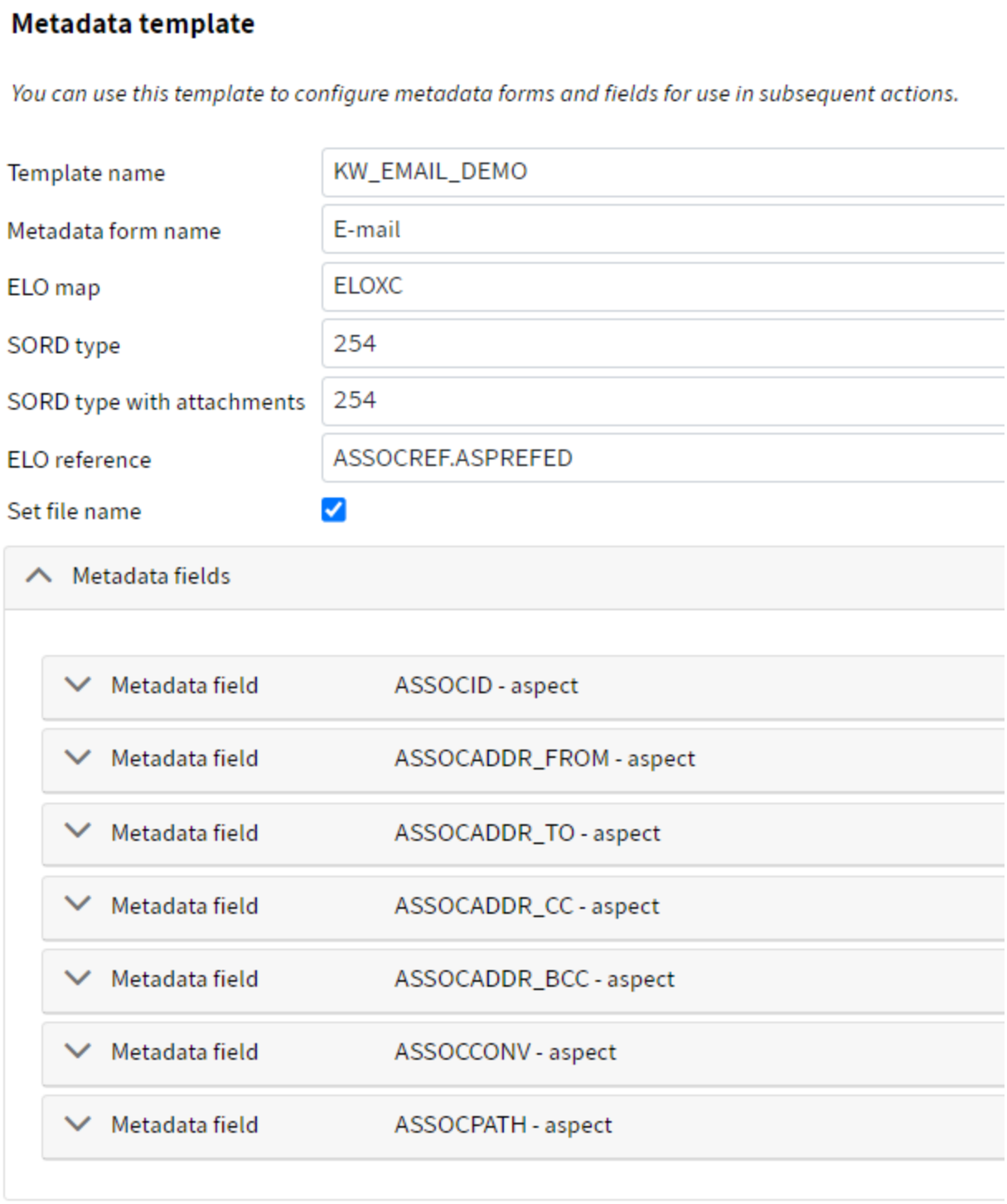


The localized display texts are stored as name-value pairs in a properties file in the */Administration/Localization/custom* folder. You can use these to configure the display texts of the forms.

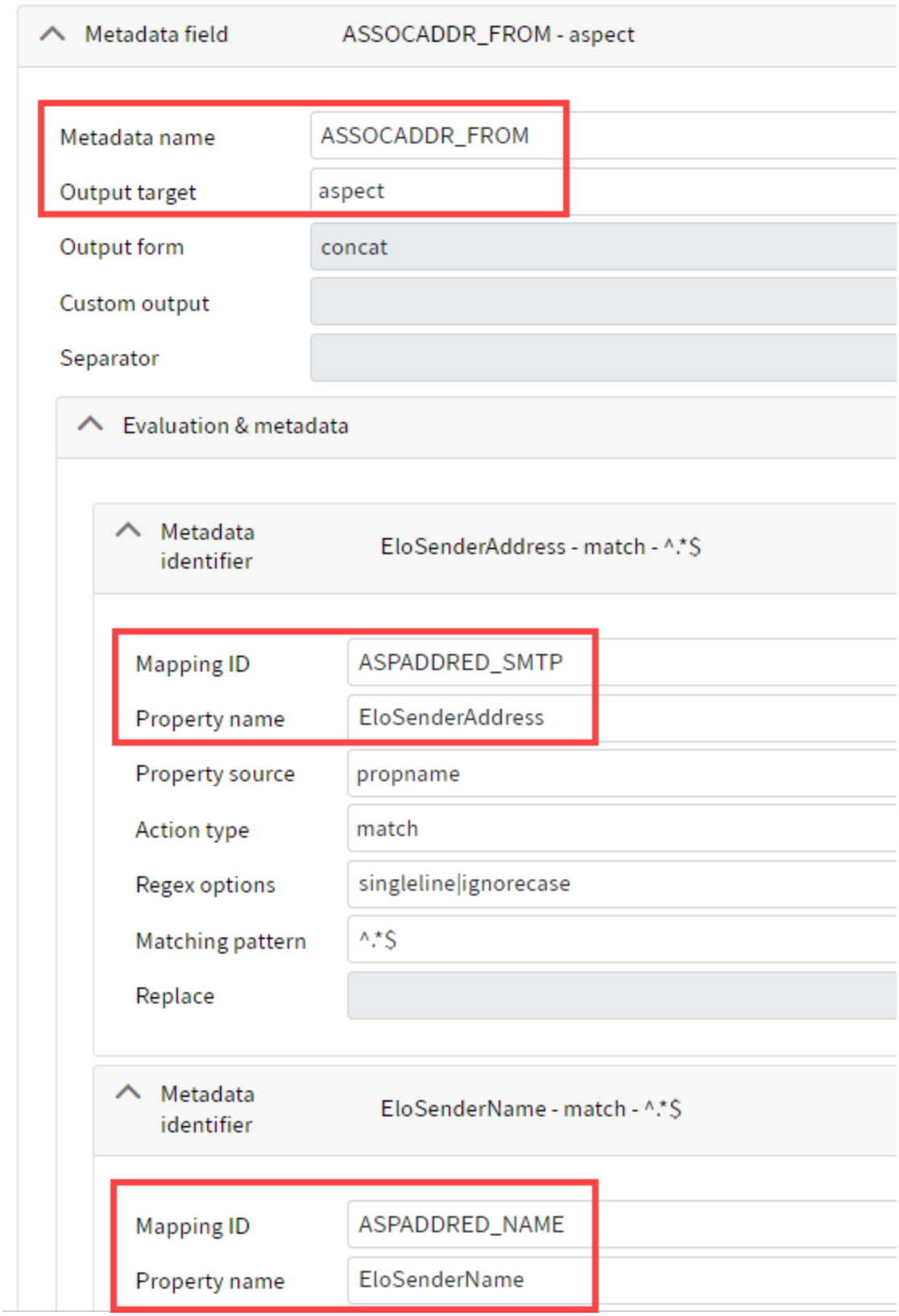


ELO XC configuration

A new metadata template for the new form needs to be created in ELO XC.



Due to the complexity of aspect forms, you need to proceed carefully here since the different levels of template configuration mean that there are more mutual dependencies than with the standard metadata forms. First, select the form and save the fragment. Next, set the metadata fields according to the aspect mappings and save the fragment again. Configure the aspect fields as metadata identifiers.



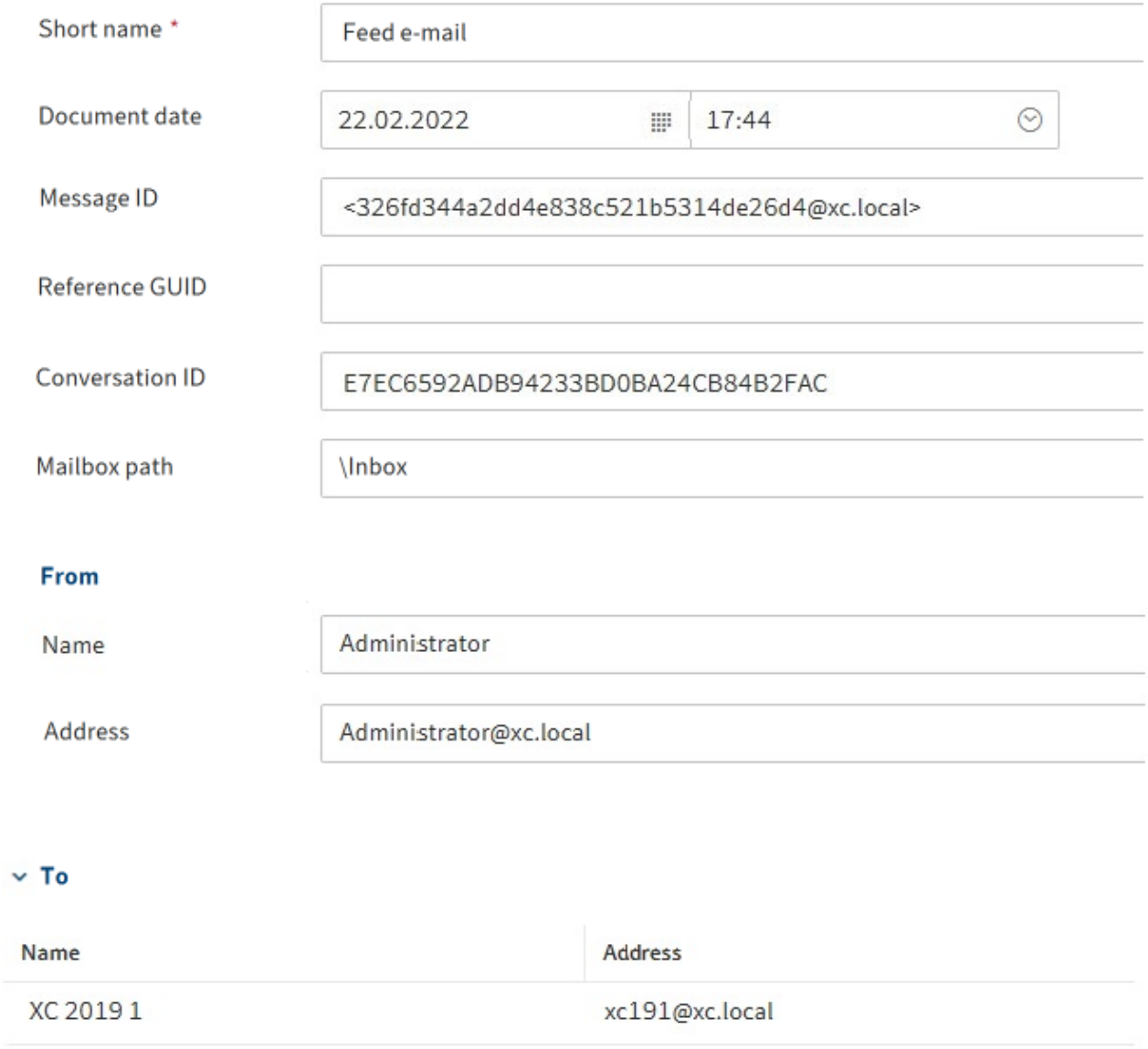
The metadata name is the aspect mapping.

The mapping ID contains the field name defined in the underlying aspect.

The property name is the Exchange field or ELO field that you want to transfer to the aspect field.

If you assign this metadata template to an export, the aspect form is used for processed e-mails.

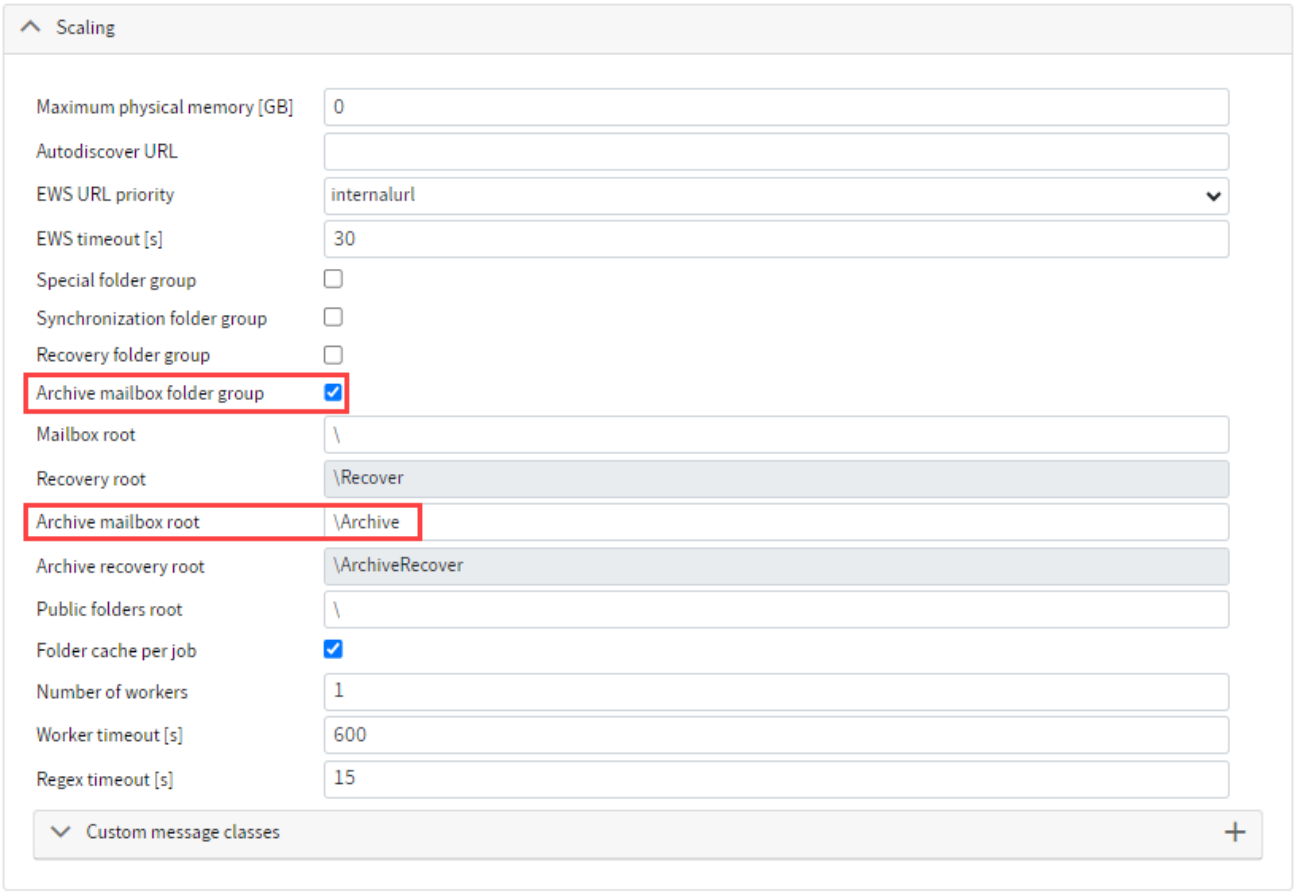
The metadata template will look like this in the client:



Processing online archives

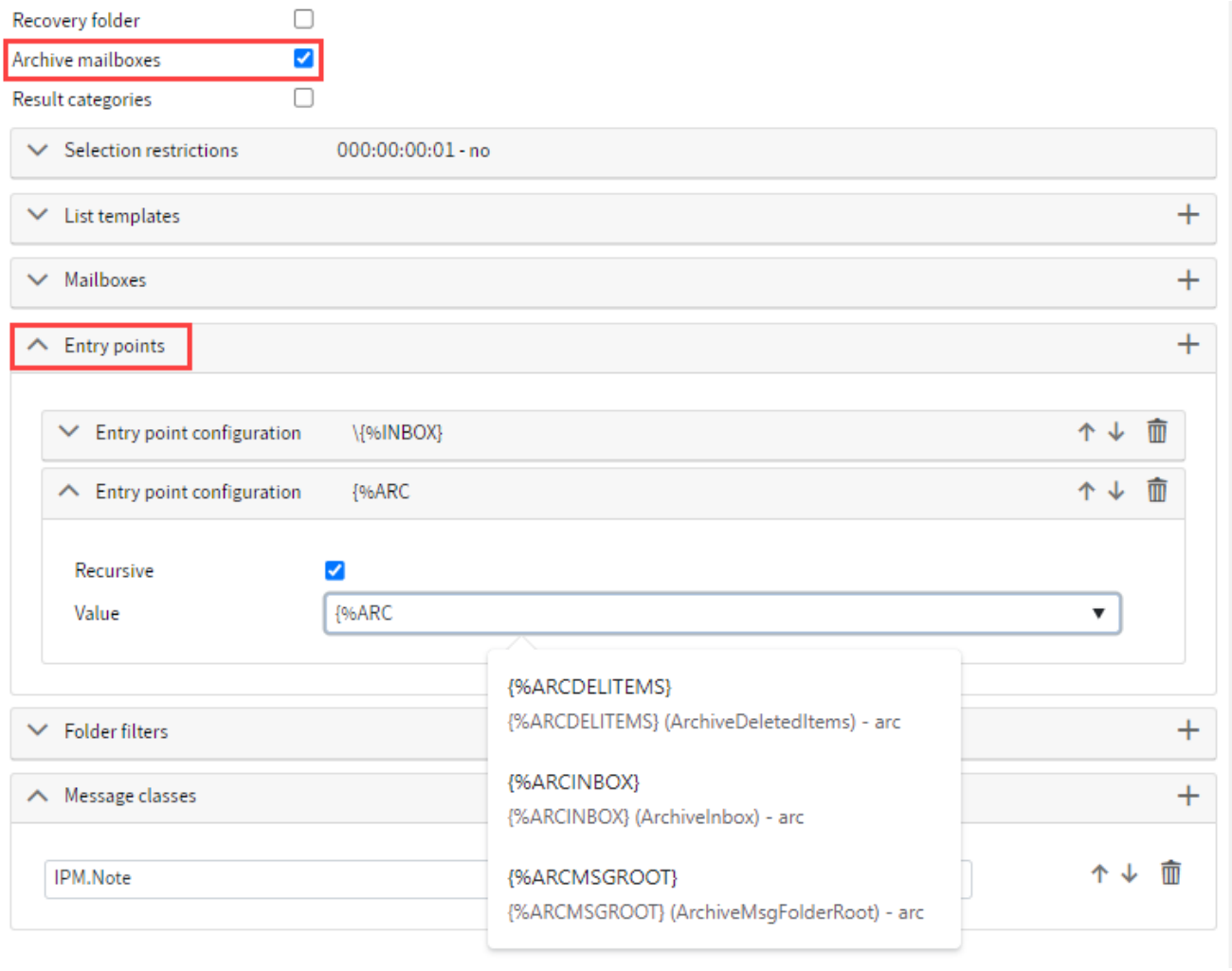
In the Exchange admin center (EAC), you can enable online archives in Exchange mailboxes. Technically, these are secondary mailboxes that are automatically available after successful mailbox authentication. Access to the contents of these mailboxes is provided based on the known folders managed by Exchange. This best practice example shows you how to configure ELO XC to process these online archives.

Not every mailbox has a secondary mailbox. To allow ELO XC to resolve and search the secondary mailbox folders for content, you need to configure the instance accordingly:



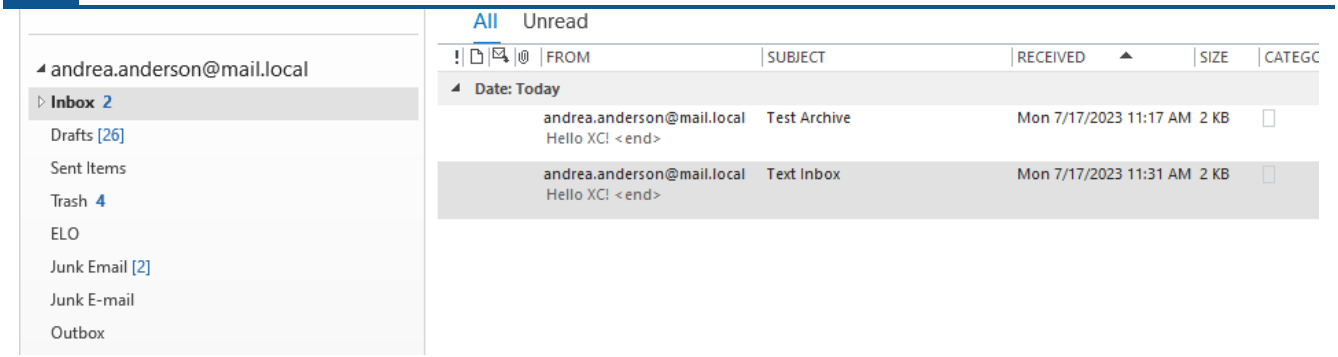
Selecting the *Archive mailbox* folder group causes ELO XC to query Exchange for folders in the secondary mailbox. The value of *Archive mailbox root* determines the display name used for the root of the determined folder structure. If the mailbox path is used to create the repository filing path, this display name is always part of all filing paths of secondary mailbox items that are processed. For example, if message A is in the inbox of the primary mailbox, and message B is in the inbox of the secondary mailbox, the mailbox paths used to store the messages will be different. A is stored in the *\Inbox* path and B is stored in *\Archive\Inbox*. If you want the same paths to be used, you can also configure ** for the root of the archive mailbox. Keep in mind that secondary mailboxes are managed by Exchange and not all known folders are automatically always available. For example, when a secondary mailbox is created with Exchange 2019, the Inbox folder is not available.

When you have finished scaling the instances, you can customize the action tree that will process archive mailboxes:

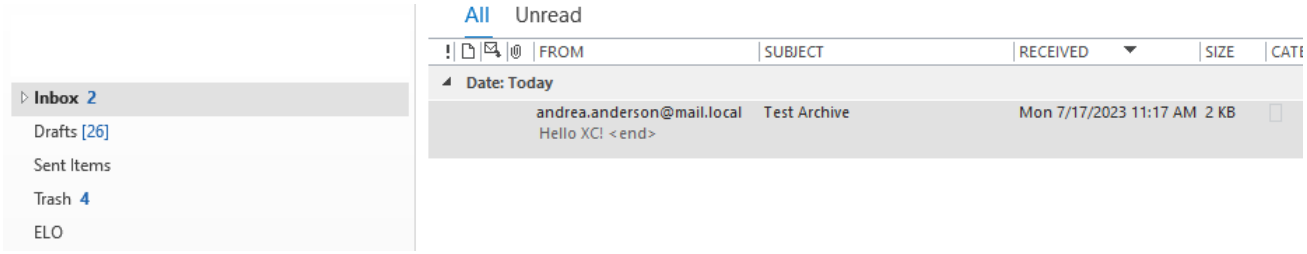


When ELO XC starts processing a mailbox, all available and configured folders are determined and an internal folder cache is built. This takes time and can be avoided by only including action trees for the folder groups that actually need them. At the same time, this means that you need to enable an action tree with the *Archive mailboxes* option in order to include the *Archive mailbox* folder group. Afterwards, the corresponding folder variables are available and are automatically suggested when configuring the entry points. In this example, select *{%ARCMSGROOT}*.

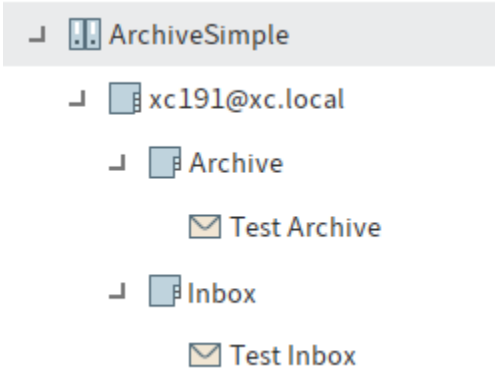
Once you have configured the filing path to use *EloBoxPath*, you can test the configuration with two test messages to the same recipient:



Move one of the messages to the online archive:



Start the instance. After the job is finished, you can find the stored messages in the repository in different paths:



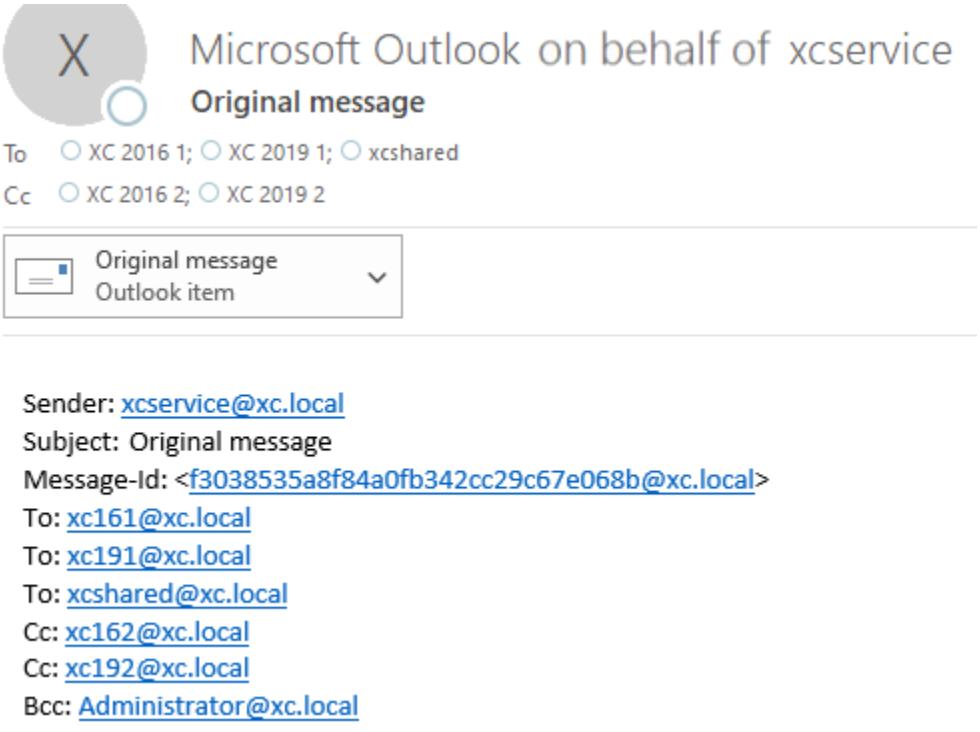
Optimizing journaling performance

ELO XC processes user mailboxes and journaling mailboxes differently. Whereas processing of user mailboxes can be extremely varied and the processing states are evident in user mailboxes, the main purpose of journaling is to process as many messages as possible in as short a time as possible. This best practice tip explains journaling, shows some examples of runtime tests to determine a suitable base configuration, and explains some more advanced configuration options for environment scaling.

Journals

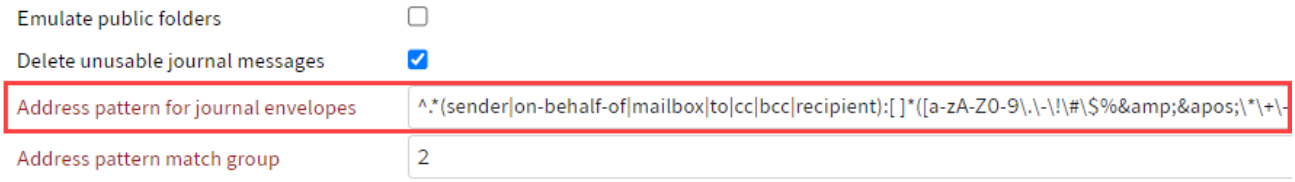
Journals are envelopes that contain original messages. Journal recipients are either configured locally for a mailbox store, or they are forwarded using transport rules in *Microsoft 365*. Technically, journal recipients are native user mailboxes. Journaling is generally used when a complete record of messages needs to be maintained. With the tests presented here, latencies could become a second underlying motivation for using journals.

Messages are forwarded to a journal at the transport level. When transporting a message, if a sender and/or recipient is found to which the journal rule applies (that is, the same mailbox store or a mention in the transport rule), Exchange puts the original message in an envelope, writes the list of all recipients on the envelope, and sends it to the journal recipient. The journal recipient gets the messages in this format:



Reading out journal envelopes

When processing journals, ELO XC examines each envelope to determine the relevant recipients. This occurs in two steps. First, it extracts SMTP addresses with regex:



The catalog data is subsequently used to check which recipients can be used for mailbox emulation. If a mailbox catalog only contains the mailbox *xc162* as in the example above, the emulation would only occur for this mailbox. However, if the catalog contains all recipients, only the mailboxes of those mailbox stores configured for the currently processed journal recipient will be emulated. *Microsoft 365* does not have this filter because it does not include mailbox stores.

Mailbox emulation

Mailbox emulation stores the original messages for each relevant recipient as if they were being processed in the associated user mailbox. It is used for journal messages and *shared mailboxes*.

Unlike processing mailboxes directly, emulation has the advantage that the original messages do not have to be uploaded to the ELO Indexserver multiple times. Instead, an internal copy is created via *IX-API*. This reduces the volume of data that has to be transmitted.

Comparison of journals and user mailboxes

The number of messages in journals is likely to be significantly higher than in user mailboxes. At the same time, journal mailboxes are not accessed because there is no mailbox owner working productively with the mailbox. Therefore, to process journal recipients, EO XC needs be able to quickly reduce large amounts of data.

While usability often plays an important role in processing user mailboxes, the main goal in journal mailboxes is to make the most of the performance limits of the environment. In this context, usability is generally hard to reconcile with performance.

Processing time

The following tests compare the performance of direct mailbox processing and journal processing. They illustrate the basic options available in the ELO XC configuration for improving performance.

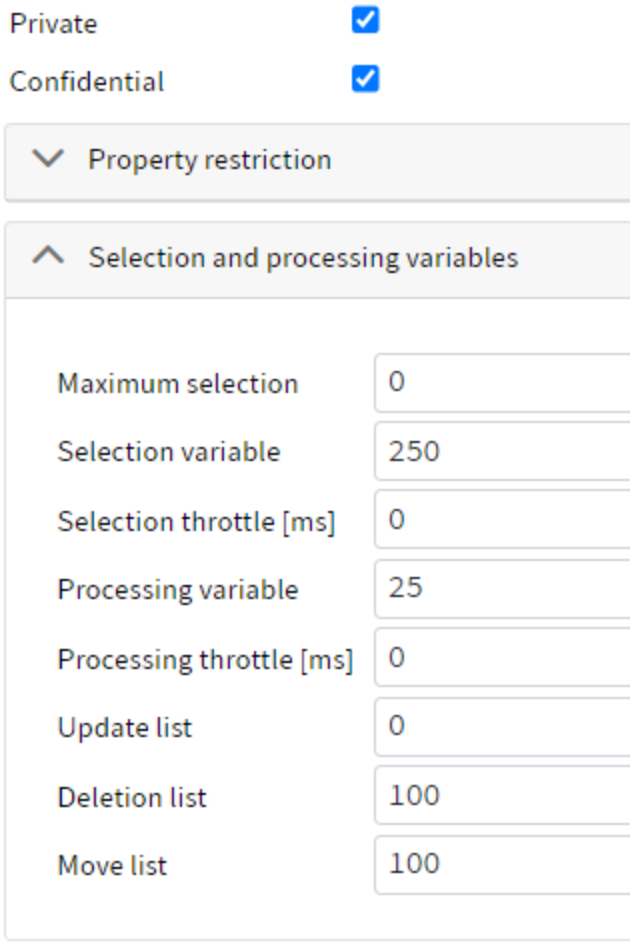
Test environment

The ELO system is a ten-year-old Windows 2012 R2 with an older MSSQL and various Tomcats (ELO servers 12, 20, and 21 with ELO Administration Console, ELO Indexserver, ELO Web Forms Services, and ELO Web Client). The ELO 21 server also contains ELO Flows. To make it easier to compare data, the full text function and event-controlled scripts are disabled.

The Microsoft Exchange environment includes a standalone test domain and two Microsoft Exchange servers (2016, 2019) with the mailboxes *xc161*, *xc162*, *xc191*, and *xc192*. The journal is on Microsoft Exchange 2016. Exchange runs with the default throttling settings. To speed up the generation of test data, the Exchange transport services were slightly adjusted.

Based on previous tests, only three *workers* are used here, as otherwise there is a risk of processing errors due to Exchange being overloaded (`server busy` or internal errors) with several thousand items towards the end of the jobs.

These default scaling values were used:



Tests

The test series evaluates the optimization steps in processing user mailboxes and journals starting from the default configuration to the maximum optimized configuration. Standard archiving assumes that messages are archived and tagged so that they are kept in the mailbox for a while afterwards. Only a second action tree deletes archived messages with a delay.

The first optimization step assumes that messages can be deleted immediately after archiving, which is a problematic assumption in user mailboxes, but not in journals. Therefore, archiving and deletion are executed directly one after the other in this optimization step. Archiving sets the

identifier (*guid*) and deletion evaluates it (search for *guid* in the repository). Both are part of the *secure deletion* standard method. In the second step, it is assumed that after successful archiving, messages may be deleted unchecked, i.e. without *secure deletion*.

In terms of optimizing journals, these fundamental questions are important:

- Why does a message have to be stored in Exchange if no one sees it and it is going to be deleted afterwards anyway?
- Why would anyone need to search the repository for a message that has just been stored?

This leads to three different configuration in this series of tests.

A estimation of the costs for a *whole* export in user mailboxes:

1. Two action trees (double selection): *EX-Bind*, *IX-Path*, *IX-DocBegin*, *IX-Upload*, *IX-DocEnd*, *EX-Save*, *EX-Bind*, *IX-Search*, *EX-Delete*

2. One action tree (simple selection): *EX-Bind*, *IX-Path*, *IX-DocBegin*, *IX-Upload*, *IX-DocEnd*, *IX-Search*, *EX-Delete*

3. One action tree (simple selection): *EX-Bind*, *IX-Path*, *IX-DocBegin*, *IX-Upload*, *IX-DocEnd*, *EX-Delete*

There is also the distinction between exporting messages as a whole or if the attachments are extracted and filed separately. With the export type *splitattachments*, the cost of storage through the Indexserver multiplies with the number of message parts, and the cost for the filing path doubles (once for the message and once for the file attachments). With emulation, the cost associated with storage through the Indexserver is drastically reduced from the second addressee onwards, because after filing the first message copy, all other copies are created by copying.

> **Please note**
>
> The *IX-Path* cost item can have a significant far-reaching effects. For example, user paths with an underlying *seconds path*, where all parts of a date (year, month, day, hour, minute, seconds) are used to create the path. Statistically speaking, almost every message would therefore have a unique filing path that has to be generated via *IX-API*. In this series of tests, paths are used that make a maximum distinction between user and message part (message or file attachment).

The tests use these combinations in the configuration settings:

Test	Type	Export	Archive	Tag	Save	Delete	secured
15	User	whole	X	X	X		
16	Journal	whole	X	X	X		
17	All					X	X
18	User	splitattachments	X	X	X		
19	Journal	splitattachments	X	X	X		
20	All					X	X
21	User	whole	X	X		X	X
22	Journal	whole	X	X		X	X
23	User	splitattachments	X	X		X	X
24	Journal	splitattachments	X	X		X	X
25	User	whole	X			X	
26	Journal	whole	X			X	
27	User	splitattachments	X			X	
28	Journal	splitattachments	X			X	

- Configuration 1: Line 15 - 20
- Configuration 2: Line 21 - 24
- Configuration 3: Line 25 - 28

Points of measurement and potential impairments

The test duration is always the difference between job end and job start. The costs associated with setting up ELO XC (e.g. Autodiscover, authentication, cache updates, determination of default folders) are included in the measurement as an unspecific constant.

The respective current size in the ELO database can vary depending on the test. A size below the range of one million SORDs is considered negligible.

Storage-only operations in ELO XC (specifically extraction of attachments) or more complex actions such as *Match/Replace* and internal evaluations of the filing path cache are considered marginal and are not taken into account.

Creation of the test data and test runs took place over several days, during which the basic performance of the entire infrastructure, particularly network load, varied. These potential impairments are also considered negligible.

Since mailbox emulation is optimized for storage of multiple documents, the corresponding volumes are not actually considered as transferred from ELO XC to the ELO Indexserver. Instead, the volumes correspond to the user data effectively generated in the physical filing paths.

Test data

For each test, 1,000 programmatically generated identical messages with attachments were sent to the four user mailboxes (body size three KB and six attachments totaling 267 KB). The journal messages are automatically generated in the Exchange environment. The overall sizes show

differences that can be attributed to differences in the internal structure of the MIME data (e.g., routes, headers, times, MIME part IDs). In contrast, the e-mail payload data stays the same.

Results for configuration 1

	Test 15	Test 16	Test 17	Test 18	Test 19	Test 20
	"whole"			"splitattachments"		
	User	Journal	Delete	User	Journal	Delete
Total time [s]	525	327	116	1211	628	116
Items	4000	1000	5000	4000	1000	5000
Item transfer rate [/s]	7.6	3.1	43.1	3.3	1.6	43.1
Documents	4000	4000	0	28000	28000	0
Document transfer rate [/s]	7.6	12.2	0	23.1	44.6	0
Volume [KB]	1,514,686	1,513,728	0	2,608,898	2,607,940	0
Data transfer rate [KB/s]	2,885	4,629	0	2,154	4,153	0

Results for configuration 2

	Test 21	Test 22	Test 23	Test 24
	"whole"		"splitattachments"	
	User	Journal	User	Journal
Total time [s]	400	302	1135	634
Items	4000	1000	4000	1000
Item transfer rate [/s]	10.0	43.3	3.5	1.6
Documents	4000	4000	28000	28000
Document transfer rate [/s]	10.0	13.2	24.7	44.2
Volume [KB]	1,514,681	1,513,723	2,608,898	2,607,935
Data transfer rate [KB/s]	3,787	5,012	2,299	4,113

Results for configuration 3

	Test 21	Test 22	Test 23	Test 24
	"whole"		"splitattachments"	
	User	Journal	User	Journal
Total time [s]	382	302	1,105	634
Items	4000	1000	4000	1000
Item transfer rate [/s]	10.5	3.4	3.6	1.6
Documents	4000	4000	28000	28000
Document transfer rate [/s]	10.5	13.5	25.3	44.2
Volume [KB]	1,514,681	1,513,723	2,608,898	2,607,935
Data transfer rate [KB/s]	3,965	5,097	2,361	4,113

Cumulative comparison

If all results according to the basic configurations are combined to show values for the use of mailboxes with specific actions in practice (combination of user mailboxes and journals), the following comparison data emerges:

	A1	A2	A3	B1	B2	B3
	15/16/17	21/22	25/26	18/19/20	23/24	27/28
Total time [s]	968	702	679	1955	1769	1739
Items	5000	5000	5000	5000	5000	5000
Item transfer rate [/s]	5.2	7.1	7.4	2.6	2.8	2.9
Documents	8000	8000	8000	56000	56000	56000
Document transfer rate [/s]	8.3	11.4	11.8	28.6	31.7	32.2
Volume [KB]	3,028,414	3,028,404	3,028,404	5,216,838	5,216,828	5,216,828
Data transfer rate [KB/s]	2,885	4,314	4,460	2,668	2,949	3,000

Overall result

The more the configurations are optimized, the higher the performance. The dramatic performance increase in configuration 2 compared to configuration 1 indicates that storage in Exchange is very expensive. In contrast, the small increase in configuration 3 compared to configuration 2 indicates that there is little difference between secure and insecure deletion, meaning that the Indexserver costs are low at this point.

The comparison between *A* and *B* also shows that there is no linear relationship between *whole* export and the *splitattachments* export type. If you were archiving seven times the number of items and the data volume doubled, the data flow would not be reduced sevenfold, but would be three times higher. This confirms the assumption that Exchange costs are significantly higher than those of the Indexserver, especially since additional Indexserver-side optimizations can be used for multiple filing sequences.

The highest performance measurements for *A* and *B* were obtained in tests 26 and 28 for journal emulation with immediate deletion. The comparison between test 26 with 22 and test 28 with 24 clearly shows that insecure deletion is not relevant. However, it must be taken into account that only ELO XC generated load on the ELO server.

Irrespective of these results, simple archiving of a single mailbox, with no extraction, no emulation, and regardless of whether it was a user mailbox or a journal, has the lowest processing time (25 percent of the costs of test 25). For qualitative testing, this optimized processing variant must be discounted because it also produces the least benefit and is trivial.

An Indexserver with few running services offers significantly higher performance than an Exchange server under the same conditions. Conversely, this means that improving the performance of the Exchange environment (e.g. transport, throttling, hardware) is preferable to optimizing the ELO server. Nevertheless, it is advisable to optimize performance of both environments and avoid suboptimal use of the ELO server resources.

Optimizing journal archiving

The ideal configuration for processing journal mailboxes is one that natively processes journal recipients as if they were a user mailbox, in other words, without emulating addressee mailboxes or extracting attachments.

The highest performance when processing journal mailboxes is achieved while maximizing the benefits.

In any case, you should avoid unnecessary write operations in Exchange. When processing journals, there is also no need to change messages and save these changes. Specific project requirements that call for greater usability may require a compromise or environment scaling.

It is much less expensive to use journaling to store messages by user in the repository than to natively process the associated user mailboxes.

Consequences

Two different use cases arise as a consequence of journal archiving:

- If journaling is used to meet compliance requirements, native processing of journal mailboxes is ideal and the resulting advantages are limited to the compliance aspect. Usability is irrelevant in this case.
- If journaling is needed to store messages by user in the repository, standard journaling with mailbox emulation is preferable to processing user mailboxes and has the highest performance in terms of compliance and user support.

Journals and 'Microsoft 365' (premium journaling)

Journal recipients for *Microsoft 365* are known as premium journaling. Their recipients cannot be configured through mailbox stores, but must be set up as transport rules to mailboxes outside of *Microsoft 365*. The following test results compare the method of processing user mailboxes in *Microsoft 365* with ELO XC and natively processing an on-premises Exchange user mailbox as premium journaling.

Configurations used:

1. Standard archiving and save (action tree 1); secure deletion (action tree 2)
2. Standard archiving and immediate secure deletion (single action tree)
3. Standard archiving and immediate insecure deletion (single action tree)

The test data was generated programmatically and in the same way as in the previous tests.

Results for the 'whole' export type

	Test 29	Test 30	Test 31	Test 32	Test 33	Test 34
	1		2		3	
	User	PJournal	User	PJournal	User	PJournal
Total time [s]	1261	288	735	227	741	216
Items	4000	1000	4000	1000	4000	1000
Item transfer rate [/s]	3.2	3.5	5.4	4.4	5.4	4.6
Documents	4000	4000	4000	4000	4000	4000
Document transfer rate [/s]	3.2	13.9	5.4	17.6	5.4	18.5
Volume [KB]	1,514,462	1,612,792	1,540,306	1,612,792	1,504,619	1,612,779
Data transfer rate [KB/s]	1,222	5,600	2,096	7,105	2,031	7,467

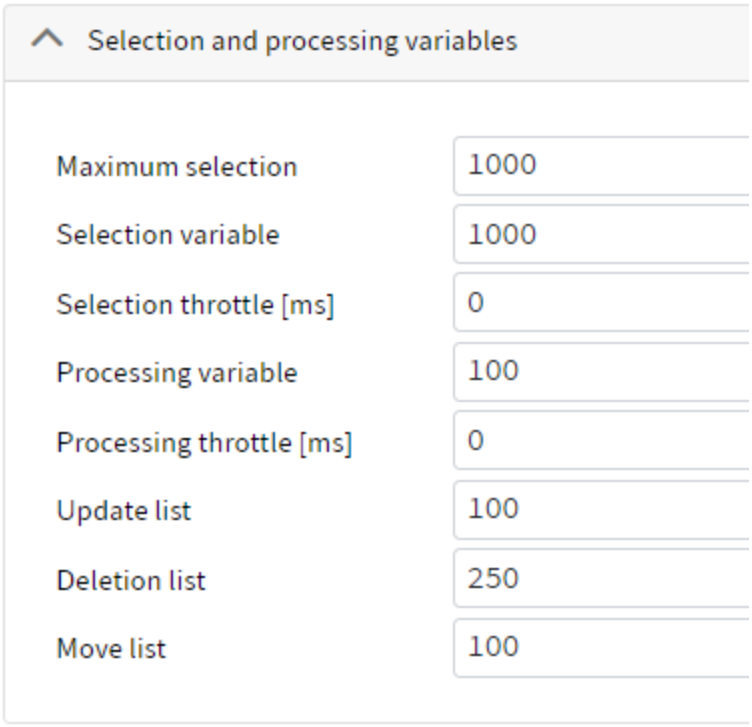
	Test 41	Test 42	Test 43	Test 44	Test 45	Test 46
	1		2		3	
	User	PJournal	User	PJournal	User	PJournal
Total time [s]	1878	622	1,402	502	1,363	501
Items	4000	1000	4000	1000	4000	1000
Item transfer rate [/s]	2.1	1.6	2.9	2.0	2.9	2.0
Documents	28000	28000	28000	28000	28000	28000
Document transfer rate [/s]	14.9	45.0	20.0	55.8	20.5	55.9
Volume [KB]	2,573,965	2,644,058	2,573,994	2,644,058	2,574,028	2,536,609
Data transfer rate [KB/s]	1,371	4,250	1,836	5,267	1,889	5,063

Optimizing parallel operation

Because it takes significantly less time to process a premium journal than when directly processing *Microsoft 365* mailboxes, it is worthwhile in every *Microsoft 365* archiving use case to replace the method of processing user mailboxes with a locally stored premium journal.

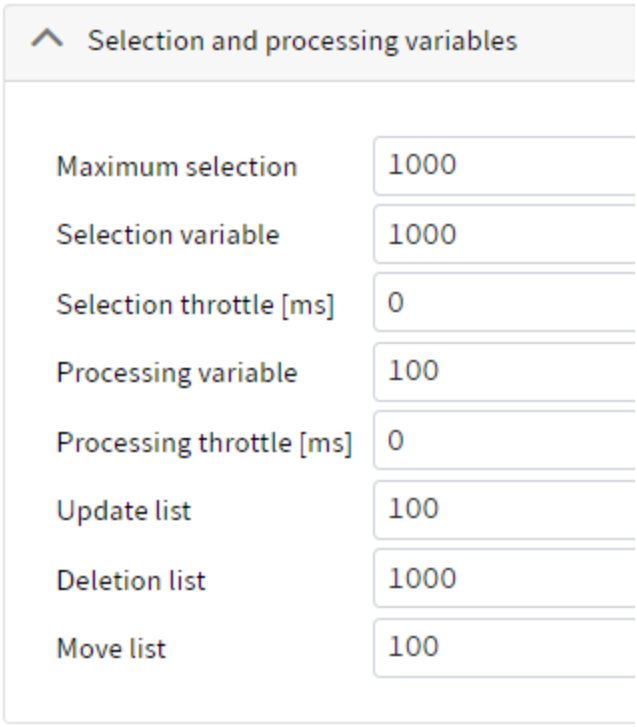
A second less obvious advantage of premium journals is that *Microsoft 365* is throttled the most. Due to this fact, all tests were performed with three *workers*, a selection variable of `250`, a processing variable of `25` and internal list lengths of `50` (*write*) and `100` (*delete/move*). These scaling values were therefore also used as standard parameters when creating new action trees in the ELO XC configuration. Tests with five *workers*, the selection variable `500`, and the processing variable `100` failed in the fourth user mailbox with the error type `Server busy`.

Local Exchange servers allow for more flexibility. A test with a local premium journal, five *workers*, and these scaling parameters resulted in processing time of 416 seconds for the export type *splitattachments*, an item transfer rate of 2.4/s (plus 20 percent compared to test 40), and a data transfer rate of 6355 KB/s (plus 26 percent compared to test 40).



With the maximum of 10 *workers*, the processing time was 419 seconds compared to the default scaling parameters in test 40, which is not an improvement compared to 5 *workers*. This is most likely due to the performance limit of the ten-year-old ELO test server. The document transfer rate with a processing time of 419 seconds is still about 67/s. If the *whole* export type is used, as in test 34, Exchange becomes overloaded quite quickly. This results in the `Server busy` error and similar errors.

A workaround for the fast journaling method (without *splitattachments*) can be tested using the Exchange IMAP interface. With the scaling factors in this test, the 1000 messages were assigned to 10 *workers* and the deletion list was only sent to Exchange after processing all messages due to its size:



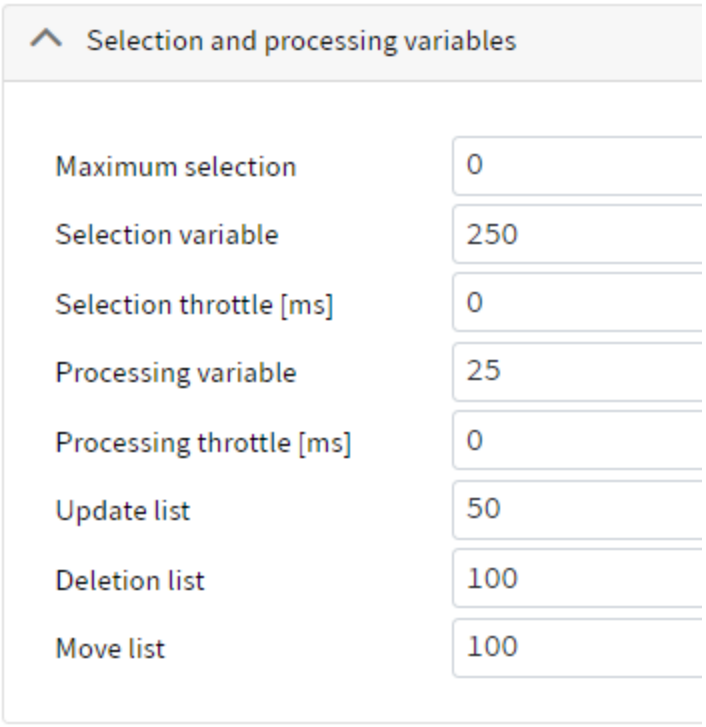
The processing time of 175 seconds has a significant impact on RAM usage, since ELO XC loads all messages for IMAP into the main memory during selection. The item transfer rate is 5.7/s and the data transfer rate is 9004 KB/s. Increasing the processing volume to 10000 journal messages results in a processing time of 1773 seconds; at ten times the volume, this is about ten times the duration plus ten times the volume of the other parameters (10000 items, 40000 documents, 15,757,548 KB). Compared to the standard scaling values used in test 34, the item transfer rate increases by 23 percent and the data transfer rate by 21 percent. Although these are good values, they are not among the top values achieved in all tests. The biggest advantage that journaling via IMAP offers is the absence of EWS throttling. If you have the option to disable EWS throttling completely, EWS is always the preferred method.

Very large messages

The majority of normal e-mails are smaller than 250 KB. If you frequently have to archive large e-mails, it is worthwhile to use journaling. The following tests were performed for 1000 messages with a size of 1.6 MB (two attachments of 1.1 MB and 0.5 MB each). All three configurations were again tested with the export type *whole*.

Failed test 1

With these scaling parameters, the messages could not be processed:



Test 41 aborted at approximately 450 items in the first user mailbox. This was caused by the `Server busy` error message and Exchange notifying the user not to try again for five minutes. This means that throttling was enabled. The problem was not that 75 messages of 1.6 MB each (3 *workers* with processing variable `25`) were loaded. The error messages occurred on saving (50 messages stacked).

Failed tests 2 and 3

Here, the update list was reduced to `10` and the number of *workers* was reduced to 1, but test 41 still failed to complete. The processing throttle configured in ELO XC should still be avoided because in practice it is equally effective for large and small items. It should only be used in emergencies.

The throttling settings of the service account were changed for this purpose. First, a policy was created:

```
New-ThrottlingPolicy -name ELOxcThrottling -RCAMaxConcurrency Unlimited -EWSMaxConcurrency
```

This policy was then assigned to a service account:

```
NSet-ThrottlingPolicyAssociation -identity xcservice -ThrottlingPolicy ELOxcThrottling
```

After that, the Exchange server *virtual machine* was restarted.

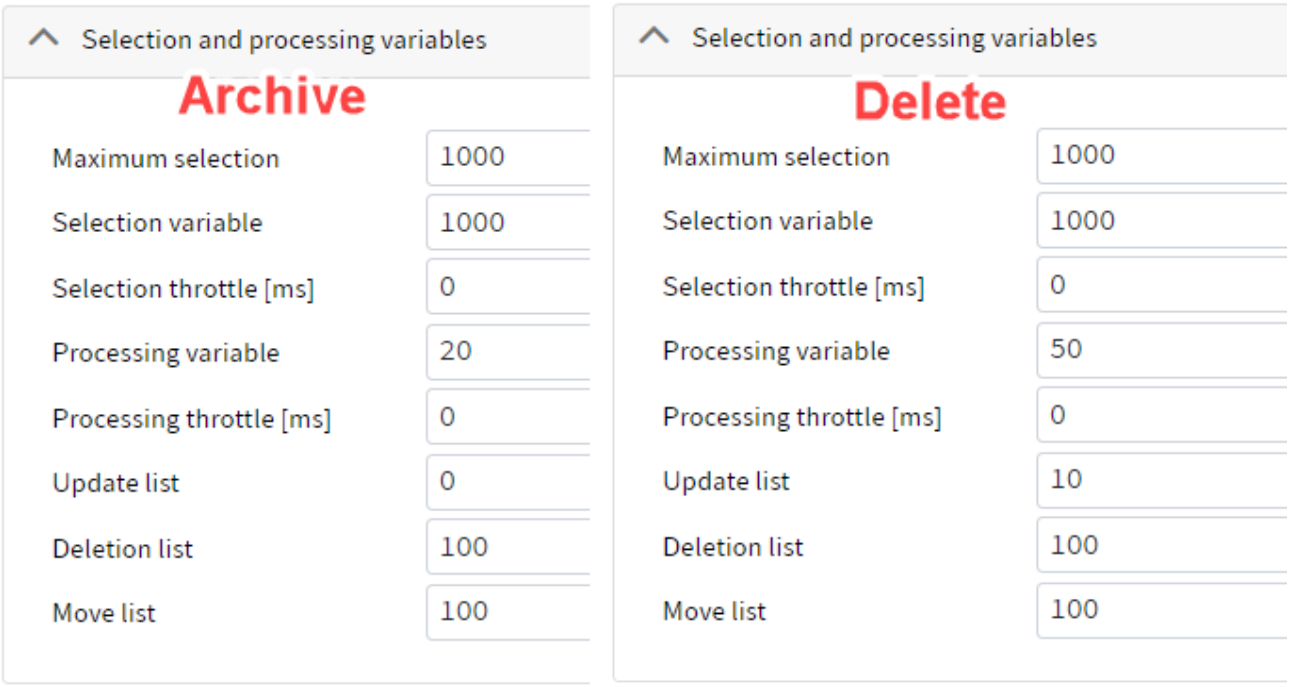
In the third attempt, about 650 messages were processed in the first mailbox, 450 in the second, and only 150 in the last two mailboxes. However, the pattern of errors changed. The error messages in Exchange changed to `The operation has timed out` or `Internal server error`.

Disabling throttling for *xcservice* was clearly not enough.

Successful test

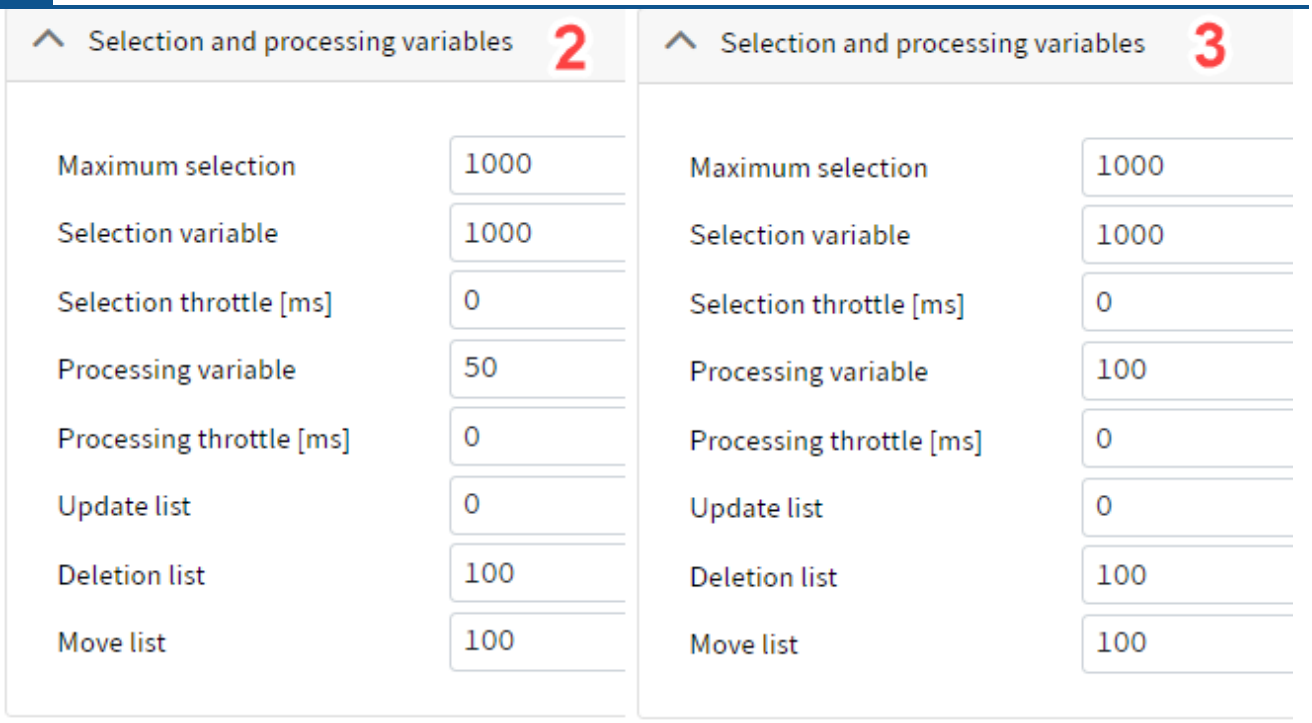
```
Get-Mailbox <mbx_filter> | Set-mailbox -ThrottlingPolicy ELOxcThrottling
```

It was only possible to process the messages in full and without errors in test 41 after applying the same throttling settings for the mailboxes.



The selection variable was maximized because only a small amount of data is transferred in the process (essentially item IDs). The processing variable of a *worker* was reduced to `20`. Batch write was disabled because it seems likely that throttling takes place in the Exchange web frontend *Internet Information Service (IIS)*. So instead of sending batches of about 16 MB to Exchange, individual messages as large as 1.6 MB were stored, which unfortunately also affected processing performance. The differences between processing user mailboxes and journaling became clearly evident in the comparison between test 41 and test 42.

Configurations 2 and 3 were tested with 3 *workers* and these parameters:



All 1000 mailbox items were once again selected. Configuration 2 simultaneously loaded 75 items from Exchange (120 MB), while configuration 3 retrieved twice that number for processing. Since neither configuration writes back items, the only relevant parameter is the deletion list. Messages not are transferred during the deletion process (hence the message size is irrelevant); only the empty item IDs are transferred. The size difference means that there is no significant effect on *IIS*, with or without throttling. Exchange can read large amounts of data within a short time.

Results for very large messages

	Test 41	Test 42	Test 43	Test 44	Test 45	Test 46
	1		2		3	
	User	PJournal	User	PJournal	User	PJournal
Total time [s]	3339	1060	610	478	585	475
Items	4000	1000	4000	1000	4000	1000
Item transfer rate [/s]	1.2	0.9	6.6	2.1	6.8	2.1
Documents	4000	4000	4000	4000	4000	4000
Document transfer rate [/s]	1.2	3.8	6.6	8.4	6.8	8.4
Volume [KB]	8,957,581	8,746,684	8,747,638	8,747,684	8,746,638	8,746,684
Data transfer rate [KB/s]	2,683	8,251	14,340	18,299	14,953	19,414

While the data transfer rate was 2 MB/s in test 33, it is almost 15 MB/s in test 45. This shows that the ELO Indexserver is well-equipped for large data volumes, while the main task of ELO XC optimization is to activate as much of the Exchange performance reserves as possible by means of the configuration.

It should be noted, however, that changing the Exchange throttling settings can have significant consequences. For example, the trade-off for processing messages in record time in ELO XC is that Exchange users will encounter mailbox access latency. Although the throttling settings can be reverted at any time by setting the global default policy for Exchange, these kinds of changes to the configuration should still be done step-by-step and with the help of the on-site Exchange administrator to ensure that the optimization is successful.

Conclusion

Journaling offers higher archiving performance on average than processing user mailboxes individually. This is already evident in on-premises Exchange servers (approx. 30 percent higher performance for the *whole* export type), but the comparison between user mailboxes in *Microsoft 365* and on-premises premium journals results in two to three times higher performance even without environment scaling.

A key factor that affects all archiving methods is the total number of write operations. With IMAP, messages cannot be changed, but requires copies of whole messages when changes are made. EWS, on the other hand, generates higher costs for write operations than for read operations when the load quota is calculated to enforce throttling settings. You should therefore always consider whether write operations are absolutely necessary. This should only rarely be the case in journal mailboxes.

Mailbox emulation offers higher performance than directly processing user mailboxes. Configurations or convenience functions designed to support users are always associated with high costs because of the write operations involved. This is particularly noticeable in the case of very large messages, which requires significant changes to the Exchange configuration as a result of these costs.

In scenarios with varying data (with/without attachments, small/large messages), it is worth allocating messages to different groups to be processed by separate action trees with different scaling values. While very small messages (less than 100 KB) may not cause any problems with an update list value of `50`, it is safe to assume that this batch size will not work for messages larger than 1 MB. Separating the action trees by message size is a good solution to the optimization problem.

Exchange costs can roughly be broken down as follows:

Read	Low
Write	High
Delete	Very low
Move	Very low