ELO server – Installation and operation

New features

Table of contents

ELO 23 basic concepts

Introduction

This document describes the changes effective from ELO 23 LTS to the server modules and the ELO Server Setup compared with ELO 20.

This documentation covers the following topics:

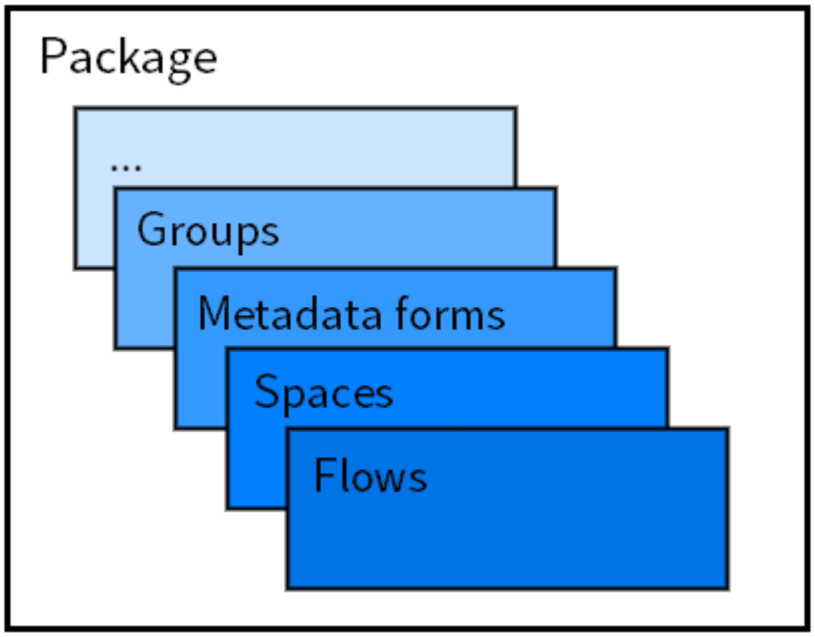
- Packages and metadata model
- Workspaces
- ELO Indexserver
- ELO Textreader (gen. 2)
- ELO XML Importer
- ELO Server Setup
- ELO database

Packages and metadata model

With the packages and the new metadata model, ELO has introduced a fundamentally new concept for managing and configuring ELO systems.

Packages

Packages allow you to create and edit related configurations. A package contains all the configurations required for a purpose.



The following elements can be embedded in packages:

- Metadata forms
- Aspects
- Fields
- Forms
- Groups
- Keyword lists
- Workspace types
- Teamspace templates
- Font colors
- Flows
- Workflows
- Translations
- Scripts
- Icons, etc.

Packages can easily be transferred to other systems. A package export contains all configurations for the available elements. They can be read into other systems and used with little effort.

The levels concept allows you to manage and update customizations separately from the default configuration.

For more information, refer to the *ELO packages* documentation.

Metadata model

The model has been extended to map metadata. An additional structure level has been introduced between the metadata forms and fields: aspects.

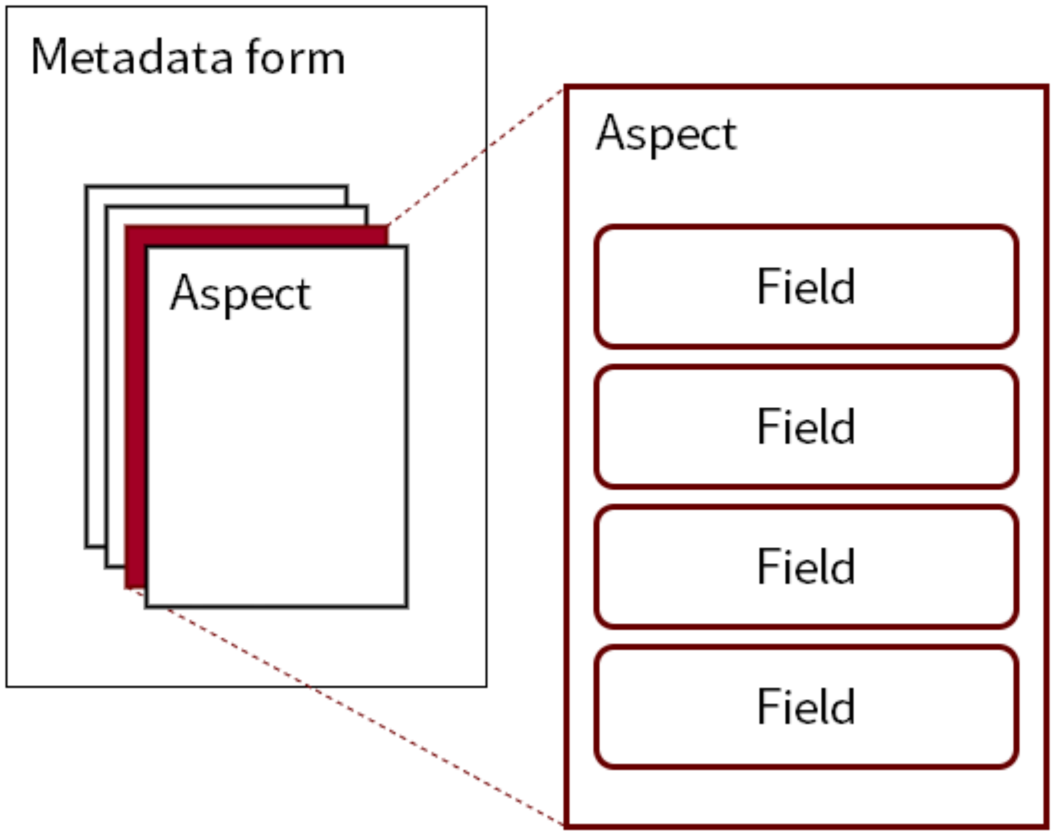
The metadata forms now have an additional data organization type: `ASPECT`. In this context, we also refer to aspect metadata forms.

For more information, refer to the *ELO packages* documentation.

Aspect forms

In simplified terms, metadata forms are made up of aspects.

The aspects contain the fields



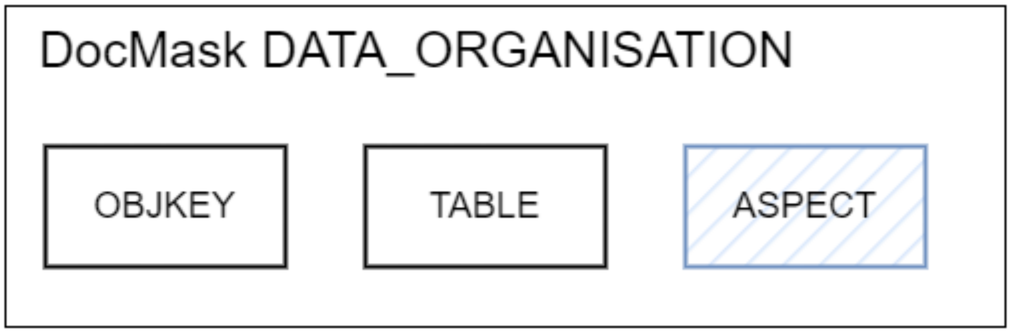
Aspect metadata forms are visualized in the interface by forms. The metadata forms no longer contain any UI-relevant information (e.g. positions of the fields).

The concept of field templates is redundant with aspect metadata forms.

The fields are labeled aspect lines in the aspect itself. Like the metadata form, an aspect is an object in the database with a unique name and ID.

Previous metadata model

The previous metadata model will be retained. The aspects and aspect metadata forms are in addition to them and are supported in parallel.



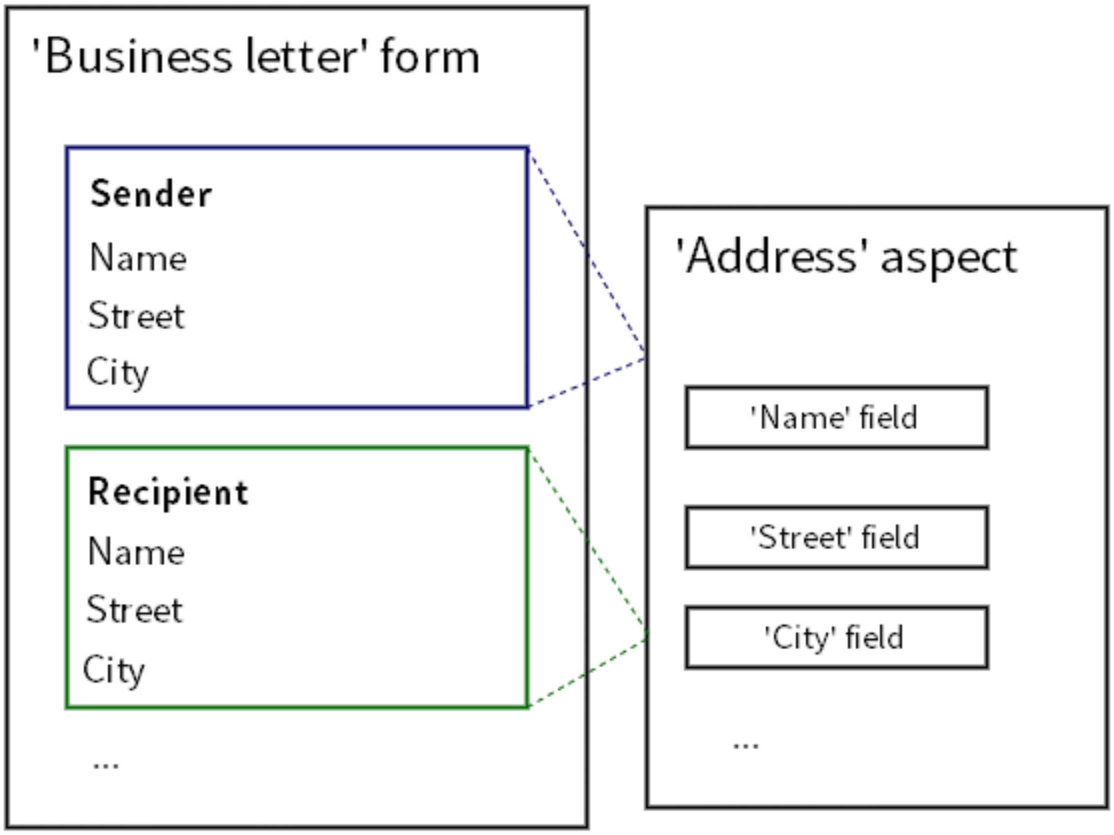
When creating a metadata form, the user chooses a data organization type for the metadata form. Later, there will not be an option to convert from/to aspect data organization.

Aspects in metadata forms – 'Aspect mappings'

A metadata form can contain an aspect multiple times – each time with a different name.

This is referred to as aspect mapping.

Example:



The *Address* aspect occurs twice in the *Correspondence* metadata form, as sender and as receiver. *Sender* and *receiver* are the aspect mappings under which the aspect occurs in the metadata form.

Aspect mapping properties

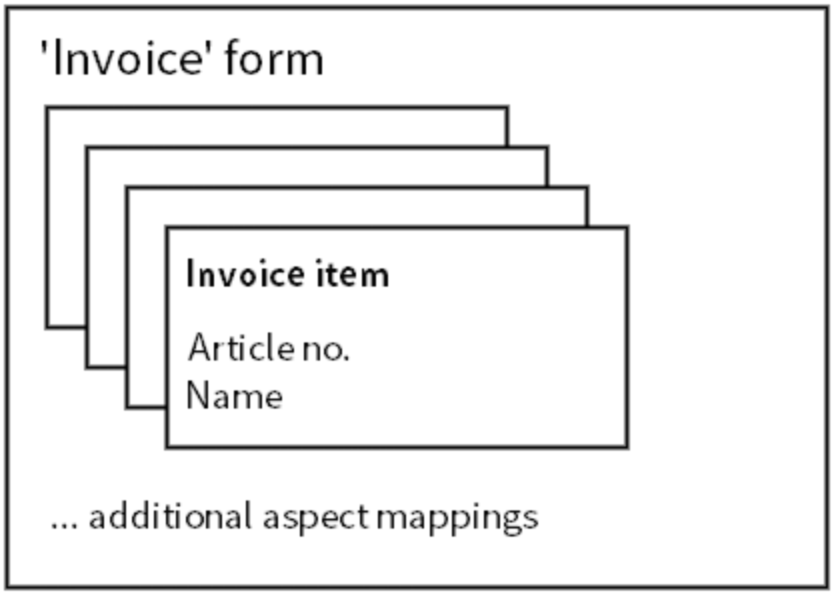
Aspect mappings are unique within the aspect metadata form via their name.

Aspect mappings are listable, i.e. they are an aggregation of the same aspects.

Each aspect mapping has a cardinality as a property:

- optional
- mandatory
- optional many
- mandatory many

Example:



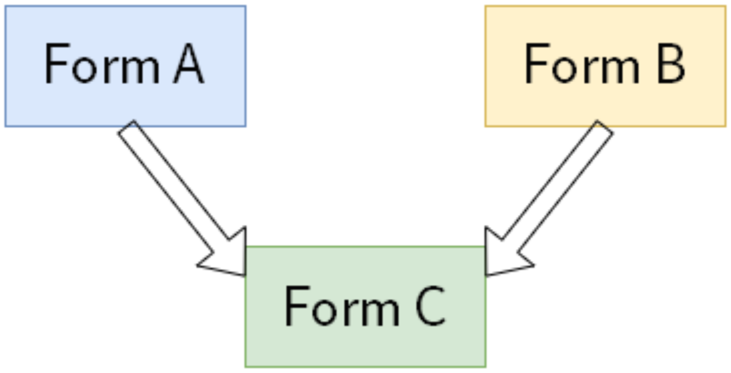
For the *Invoice item* aspect with the fields *Article no.*, *Name*, *Quantity*, and *Price*, the *Invoice* metadata form has a listable aspect mapping *Invoice item* with the cardinality *mandatory many*.

An aspect mapping is mapped in the code by the new API class `AspectAssoc`.

The cardinality is expressed by the new API class `Cardinality`.

Inheritance relationships between metadata forms

In aspect metadata forms, there can be inheritance relationships. Aspect metadata forms can "inherit" aspect mappings from other aspect metadata forms. Inheritance is additive only, i.e. inherited aspect mappings cannot be changed or overwritten in the child metadata form.



As you can see in the image, a child metadata form can inherit from multiple parent metadata forms. No contradicting properties of the inherited aspect mappings and locally defined aspect mappings may occur. This should be avoided in the concept. When checking in a metadata form, the ELO Indexserver checks whether this results in inconsistent aspect mappings or cycles in the inheritance structure.

The essential properties of the aspect mappings are:

-

The name of the aspect mapping as a unique characteristic within the metadata form
- The ID of the aspect that the aspect mapping refers to
- The cardinality of the aspect mapping
- The access rights of the aspect mapping

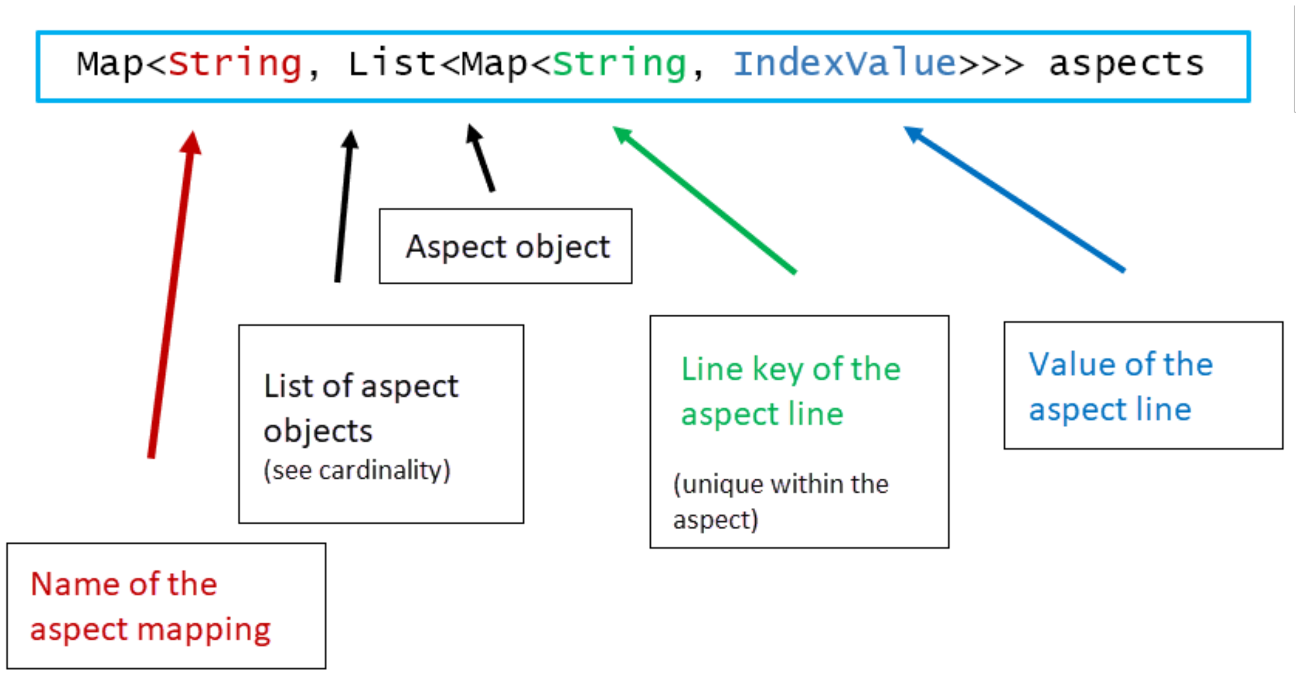
A metadata form can inherit a locally defined aspect mapping and an aspect mapping with the same name from another metadata form. However, all of the properties above have to match. The aspect allocation only occurs once in the result in the child metadata form.

A parent metadata form inherits all its aspect mappings to its child metadata form(s), i.e. both the aspect mappings you defined locally and the aspect mappings it inherited itself from other metadata forms.

Aspect objects in the 'Sord' class

After the structure of the metadata is defined via aspects and metadata forms, the concrete field values can be mapped in the aspect objects in the `Sord` class.

The aspect objects of a SORD can be found in the `aspects` structure.


Map<String, List<Map<String, IndexValue>>> aspects
Aspect object
List of aspect objects (see cardinality)
Line key of the aspect line
(unique within the aspect)
Value of the aspect line
Name of the aspect mapping

The aspect objects of a SORD are organized as a map structure:

```
Map<String, List<Map<String, IndexValue>>> aspects
```

The external map provides a list of aspect objects for a specific aspect mapping. A list as the aspect mappings have a cardinality and are listable (see section Aspect mapping properties).

The internal map returns the index value for a field. The key in the map is the defined line key of the aspect line.

Example in Java:

```java
String zip = sord.getAspects();
int quantity = sord.getAspects();
```

Information

The external map always returns a list of aspect objects, even if the cardinality of the aspect mapping is "many". The only entry is then at position 0 of the returned list.

The aspect objects of a SORD are read/written when the member flag `SordC.mbAspectObjects` is set.

Values of index fields: 'IndexValue'

For SORDs created with an aspect metadata form, values of fields are shown using the class `IndexValue`. The `IndexValue` is a serializable container for the field value and has a type.

The `IndexValue` is assigned a value depending on the aspect line type.

The value is accessed depending on the type, e.g. with `getIntValue()` etc.

An `IndexValue` is explicitly not arrayable or listable.

Field types

The type of a field is defined in the `AspectLine` class when defining an aspect. The later value of the field in the `IndexValue` class has a corresponding type.

The following aspect line types are currently supported:

- Text
- Integer
- Double
- Relation
- ISODate_ONLY (date without time)
- ISODate_TIME (date with time)
- Status

The type constants are defined in the `AspectLineC` class and based on the previous class `DocMaskLineC` constants.

The following corresponding types of the `IndexValue` class exist for the aspect line types:

Aspect line type	IndexValue type	Access to the value
TYPE_TEXT	TYPE_STRING	`getStringValue()`
TYPE_INTEGER	TYPE_INT	`getIntValue()`
TYPE_DOUBLE	TYPE_DOUBLE	`getDoubleValue()`

Aspect line type	IndexValue type	Access to the value
TYPE_RELATION	TYPE_RELATION	`getStringValue()`
TYPE_ISO_DATE_ONLY	TYPE_ISO_DATE_ONLY	`getIsoDateOnlyValue()`
TYPE_ISO_DATE_TIME	TYPE_ISO_DATE_TIME	`getIsoDateTimeValue()`
TYPE_STATUS	TYPE_STATUS	`getStringValue()`

Subtype

The aspect line contains an additional optional property: the subtype.

The subtype can be set by a client or applications in order to carry out use-specific formatting of values in the user interface. The type information for the server is separated from the database and formatting information for display in the UI.

The subtype is saved by the server but not analyzed.

The STATUS type

The STATUS aspect line type is designed for values that the user can only select from a pre-defined list of values, e.g. status information such as "Paid", "Reserved", "Valid", etc.

The fixed values are defined in a keyword list for the field.

With a status field, the `IndexValue` in the `displayData` field has a value translated into the user language as a property. This requires translated values (translation table or properties). When checking out a SORD, the display data values of the `IndexValues` class, STATUS type, are filled with the translated value. If this value does not exist, `displayData` is assigned the string value of the `IndexValues` class.

The keyword list of a type STATUS aspect line also has the translated values: there, they can be found in the `Keyword.displayValue` field.

Mapping the inheritance relationship between metadata forms

The inheritance hierarchy of the metadata forms (see section Inheritance relationships between metadata forms) is implemented by another API class: `DocMaskInherit`

The `DocMask` class is given an additional field: the list `inheritFromMasks`.

```
protected List<DocMaskInherit>inheritFromMasks;
```

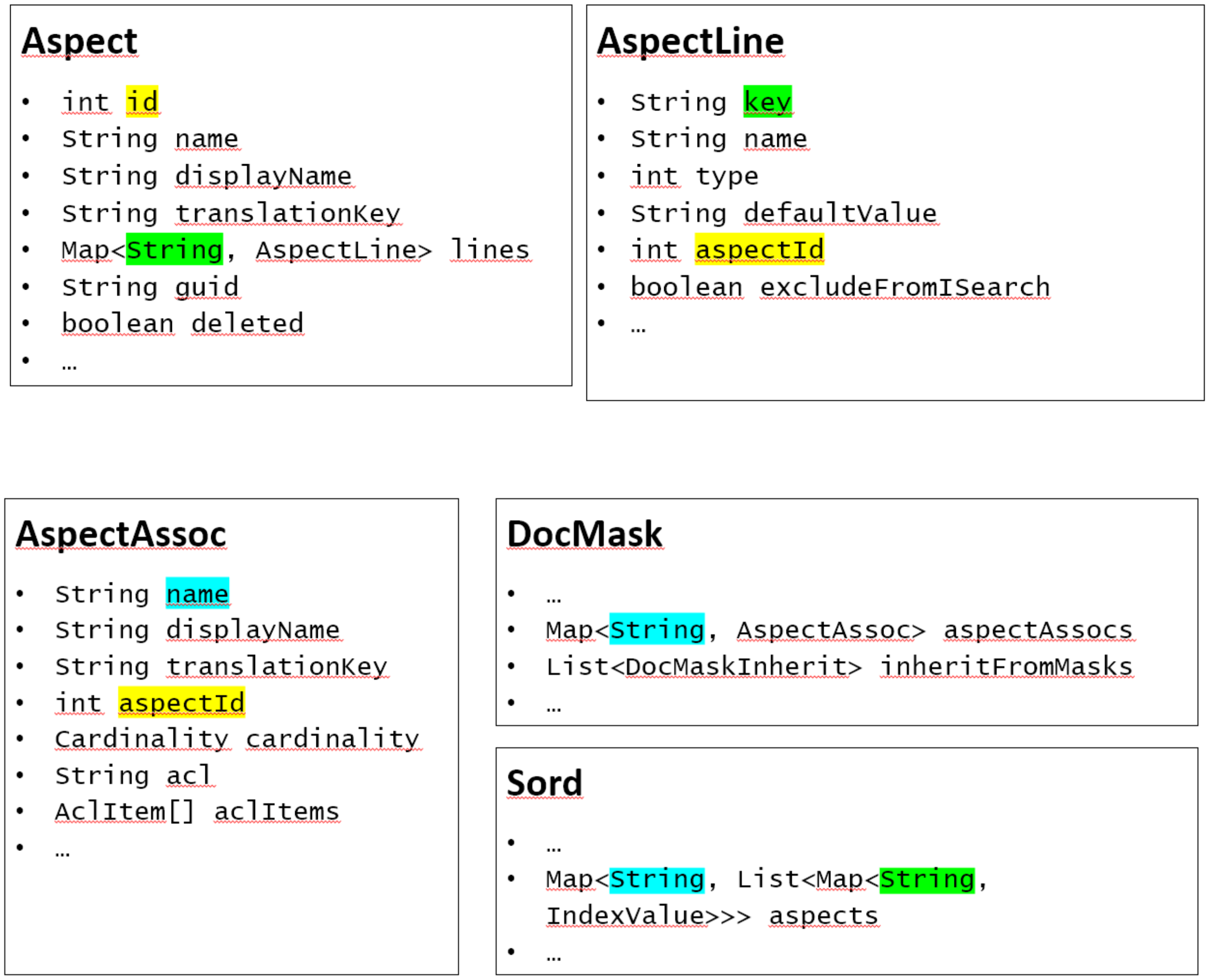
The list contains `DocMaskInherit` type objects. Each `DocMaskInherit` object contains the parent ID of a parent metadata form of the current metadata form.

So, each metadata form is "familiar" with all its direct parent metadata forms via this list.

The parent relationship for a metadata form can be set and changed as with all changes by checking in the metadata form. A separate member flag also exists in the `DocMaskC: mbInheritFromMasks` class for this.

New and changed API classes

The following addresses the most important new API classes and the most important additions to existing classes. The names highlighted in color correspond to the parts of another class with the same color.



Aspect

- `int id`
- `String name`
- `String displayName`
- `String translationKey`
- `Map<String, AspectLine> lines`
- `String guid`
- `boolean deleted`
- …

AspectLine

- `String key`
- `String name`
- `int type`
- `String defaultValue`
- `int aspectId`
- `boolean excludeFromISearch`
- …

AspectAssoc

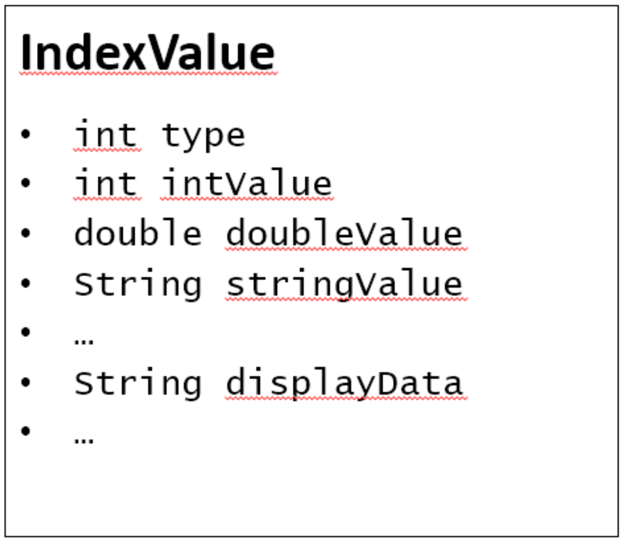
- `String name`
- `String displayName`
- `String translationKey`
- `int aspectId`
- `Cardinality cardinality`
- `String acl`
- `AclItem[] aclItems`
- …

DocMask

- …
- `Map<String, AspectAssoc> aspectAssocs`
- `List<DocMaskInherit> inheritFromMasks`
- …

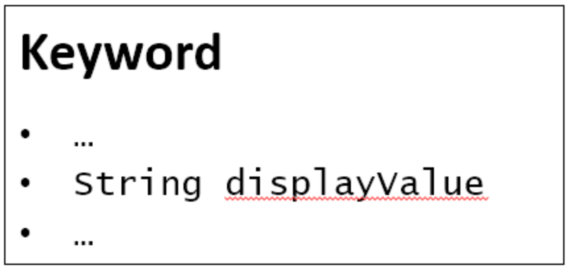
Sord

- …
- `Map<String, List<Map<String, IndexValue>>> aspects`
- …



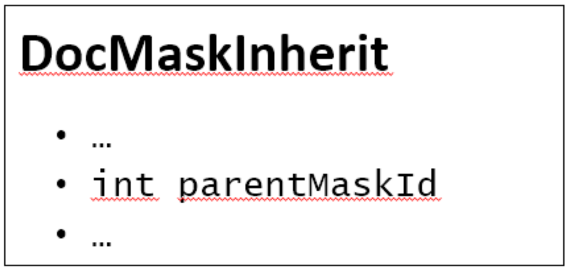
IndexValue

- int type
- int intValue
- double doubleValue
- String stringValue
- …
- String displayData
- …



Keyword

- …
- String displayValue
- …



DocMaskInherit

- …
- int parentMaskId
- …

Additional new API classes

Several additional new utility classes have been added to the API:

AspectC, AspectLineC, AspectZ, AspectInfo, Cardinality, IndexValueC

New database tables

aspects

id

name

lockid

translationkey

guid

tstamp

status

tstampsync

packagename

aspectlines

aspectid

linetype

linedisplayname

linekey

lineflags

lineext

linedefault

aspectlines

linenametrkey

linecomment

allowedrefmaskids

dynkeywordreference

subtype

linecommenttrkey

aspectassoc

maskid

name

translationkey

aspectid

cardinality

acl

docmaskinherit

parentmaskid

childmaskid

Aspect object tables

The values of the fields (`IndexValue`) are stored in the aspect object tables. This kind of table is generated dynamically for each aspect when the aspect is created.

An aspect object table is named with `asp_<aspectid>`, whereby `<aspectid>` stands for the ID of the aspect.

A row in an aspect object table saves an aspect object.

An aspect object table has the following columns:

Column name	**Type**	**Description**
objid	int	Object ID (sort ID)
assoc	varchar	Name of the aspect mapping
ordinal	int	The position of the aspect object in the list of aspect objects for aspect mapping. Always 0 for not-many cardinality.
regionobjid	int	Reserved for extensions
maskId	int	ID of the Sord metadata form.
tstamp	varchar	Timestamp of last change
Index value		One value for each field. The type is determined by the field.

Important general information

Reserved fields

The previously reserved fields (e.g. `ELO_FNAME`) initially remain stored in the structure of the previous ObjKeys. So, if a SORD is generated with a metadata form, it still contains ObjKeys. These ObjKeys may only contain the contents of the reserved fields, but no "normal" index values for regular fields.

Changes on the search API for the iSearch

Preliminary remarks

To uniquely reference a field within a metadata form index ini the ELO iSearch in a `QueryFilter`, previously you only had to specify the key (`LineKey`) of the metadata form line (`DocMaskLine`).

For metadata forms with aspect data organization, there is also the name of the respective aspect mapping.

Create QueryFilter

Until now, the field within the metadata form that the filter applied to was specified in the `docMaskLine` of the `QueryFilter`. This field is now marked as deprecated.

This has been standardized and the `indexFieldKey` field is now used for previous metadata forms with `ObjKey` data organization and for metadata forms with aspect data organization. The previous field `docMaskLine` is now only evaluated as an alternative if `indexFieldKey` is empty, and only for non-aspect metadata forms.

For non-aspect metadata forms, the `lineKey` for the index field still has to be entered. For aspect metadata form fields, the name of the aspect mapping is followed by the separator "¶", followed by the `lineKey` of the aspect line, i.e. `assocName¶lineKey`.

Example:

```
CUSTOMER ADDRESS¶LOCATION
```

Field name within Elasticsearch

The schema of the field names in Elasticsearch was retained. For example, for a tokenized input field: `LINE_<indexFieldKey>.tokenized`

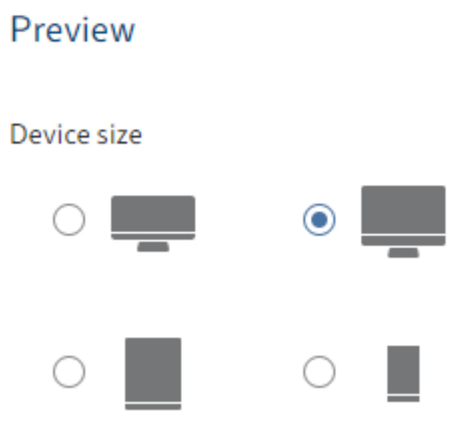
Example:

```
LINE_CUSTOMER ADDRESS¶LOCATION.tokenized
```

Forms (gen. 2)

The change to the new metadata model will also be accompanied by a new generation of forms.

The new generation of forms is characterized by a more flexible design concept. Current web design standards replace the inflexible model of table layouts.



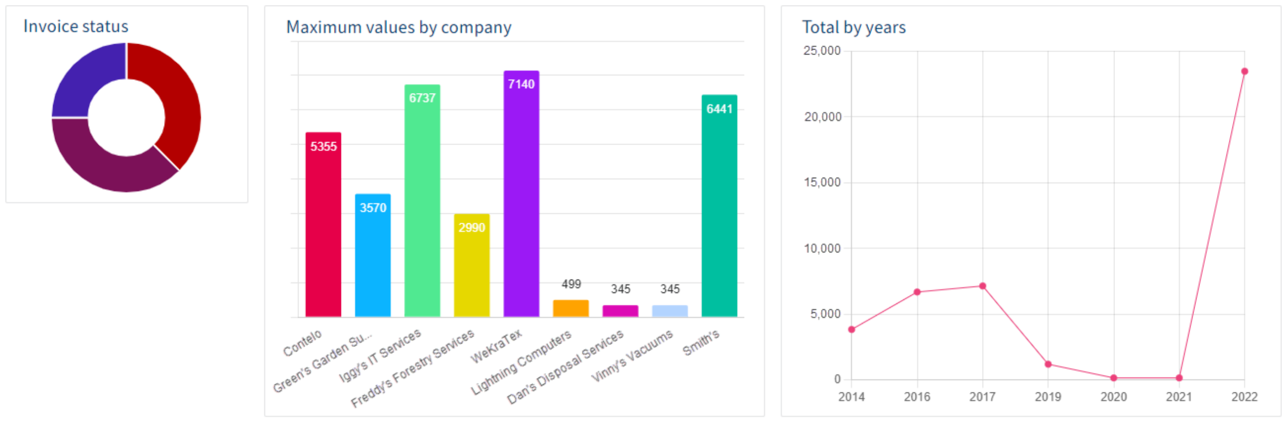
Thanks to responsive design, forms can be adapted and displayed on different devices.

The new form designer has a fully functional preview.

For more information, refer to the *ELO packages* documentation.

Dashboards and ELO Analytics

A technical change to the Elasticsearch technology has made it impossible to continue operating ELO Analytics.



ELO now offers dashboards instead. These are based on the forms (gen. 2) and can be embedded into workspaces as preview elements. They can also be shown in the clients when performing a search if the respective metadata form has been configured accordingly.

Dashboards are configured via the metadata forms.

For more information, refer to the following sections of the *ELO packages* documentation:

Workspaces

A workspace is a special area in ELO where the rights are automatically assigned for work groups.

Teamspaces can contain a workspace.

'Sord.spaceGuid' field

A workspace is essentially an ELO folder with a dedicated flag: `sord.details.workspace = true`.

The new property `sord.spaceGuid` is set for the child elements of this folder. The value of this property is the GUID of the respective workspace where the SORD is located: `sord.spaceGuid = workspaceSord.guid`. This field `spaceGuid` is not set for the objects that do not belong to any workspace. The property `spaceGuid` is not set with its own GUID for the workspace folder, as it can be located in a teamspace. This property is automatically set in the background when creating and moving an object and cannot be edited manually. During a check-in operation, even with the right selector `SordC.mbSpaceGuid`, a modification to the `spaceGuid` property is simply ignored.

X right

The X right has been implemented as a right for workspaces in addition to the RWDELP rights.

If a user group with the "X" right is added to a child folder in the workspace, the system ensures that this group is linked to all groups of the workspace with an AND in terms of permissions. Only members of the group with the X right have access to the object.

If another group without the X right is assigned, the folder can be read by all members of this group. In this case, no special AND is formed.

With this kind of concept, you no longer have to concern yourself with the AND in reference to workspace groups, but could allow an object in the workspace to break out in individual cases.

The X right for the workspace SORD itself does not have any effect. You can declare a delete right that is only applied for workspace child elements but not the workspace folder itself, for example. So we have to differentiate between two types of ACEs on the workspace folder:

- The workspace folder ACEs without the X right only apply to the workspace folder itself and are not AND-linked to the ACL of the workspace child elements.
- The workspace folder ACEs with the X right in turn do not have any significance for the workspace folder itself, but are instead AND-linked to the ACL of the workspace child elements that also have the X right.

Additional changes

ELO Indexserver

The following changes have been made to the ELO Indexserver.

Support for Tomcat 10

Apache Tomcat 10 has been supported since ELO 21.4. This occurred at the same time as the switch to JavaEE 9, which is incompatible with previous versions due to the namespace change from *javax.servlet* to *jakarta.servlet*. It is necessary to check whether these packages are being used in web applications or plug-ins that run on Apache Tomcat or ELO Indexserver, and to adapt them accordingly.

Java 17

The ELO Indexserver is now compiled with Java 17, which is now the minimum Java version. The ELO Server Setup updates the environment accordingly.

Changes to ELO Indexserver status page

The information on the status page of the ELO Indexserver and other ELO modules is now only displayed after authentication of the Tomcat account.

Without authentication, only the execution status is displayed, e.g. *Running*. Error messages are no longer displayed.

New error code for failed logon

If the session of a client application expired, until now the ELO Indexserver acknowledged the next call with the error code `IXExceptionC.NOT_FOUND = 5023`. This is the case, for example, if the connection to the LDAP server is temporarily unavailable (if configured). In this case, the client application cannot distinguish whether the sought object cannot be found or the re-logon has failed depending on the call.

A new error code was introduced for this purpose: `IXExceptionC.AUTHENTICATION_FAILED = 3008`. The ELO Indexserver now returns this code when authentication fails.

'IXConnFactory' functions deprecated

Because there are now fewer options on the status page, the information about the `readStatusPageProperty` and `readStatusPageProperties` functions is no longer available. The functions based on these are therefore deprecated.

Deprecated functions:

Function	Behavior	Alternative
`readStatusPageProperty`	Returns `null`	None
`readStatusPageProperties`	Returns empty property object	None
`getVersion`	Returns constant value `23.1.0.0`	`IXConnection.getVersion`
`getMajorVersion`	Returns constant value `23`	`IXConnection.getMajorVersion`
`getStatusPageErrors`	Returns empty array	None

Because of this change, client libraries of older ELOix versions can no longer access this ELO Indexserver.

'OnEnterHandleRollback' and 'OnExitHandleRollBack' script functions

In the workflow nodes, you now have access to two more fields with new functions for the scripts:

- `onEnterHandleRollback`
- `onExitHandleRollback`

The functions need the same parameters as the `onEnter`/`onExit` methods. To run them in JavaScript, you need to add `Node` to the function in the script.

Example:

```
function onEnterHandleRollbackNode(ci, userId, workflow, nodeId)
```

- `ci` = ClientInfo
- `userId` = The ID of the user who processed the workflow node
- `workflow` = The active workflow
- `nodeId` = Workflow node ID where the script is

These functions can only be executed if they are in the script in the field with `onEnterHandleRollback`. This is similar to the fields with `onEnter` or `onExit`.

The script in the field with `onEnterHandleRollback` will only be executed if there is also a script with `onEnter` in the same node.

The same applies for the combination `onExitHandleRollback` and `onExit`.

The functions `onEnterHandleRollback` and `onExitHandleRollback` are not designed to fix errors in the respective scripts with `onEnter` or `onExit`, because the scripts in the fields with `onEnterHandleRollback` and `onExitHandleRollback` are stored temporarily and are only executed if an error occurs when forwarding a workflow from one person node (`onExit`) to the next person node (`onEnter`).

Scripts with `onEnterHandleRollback` or `onExitHandleRollback` are only stored temporarily if the associated script with `onEnter` or `onExit` was executed successfully and without errors.

The scripts are executed in the following order:

-

The script with `onEnterHandleRollback` or `onExitHandleRollback` that was added last is executed first.
- The script with `onEnterHandleRollback` or `onExitHandleRollback` that was added first is executed last.

Check the files on upload

When uploading a document, an Intray file, or an import ZIP, you have the option to check for viruses. You can also specify filters for file extensions and content types that reject certain file types or only accept certain file types.

Information

If you use the option values described below, you do not need to restart or reload the ELO Indexserver.

Check the file content

You specify a call line for a program in the options that checks the file content when uploading a document. The `$1` placeholder in the call line is replaced with the file path.

If the program returns the `code=0`, the file is accepted. A different code will abort the upload with an `IXExceptionC.INVALID_PARAM` (`[ELOIX-2000]`) message.

The program is called with `verifyFileUploadCommand` and is specified as the instance name *_ALL* so that all ELOix instances run the check.

In addition, `verifyFileUploadTimeoutSeconds` can be used to specify the maximum time the program is allowed for a check. If the timeout is exceeded, the ELO Indexserver kills the check program and returns an error message `ELOIX-2000`. The default timeout value is 30 seconds.

Example:

```
clamdscan --fdpass $1
```

Check the file type

In addition to checking the file content, you also have the option to check the file extension, i.e. content type. The `allowedFileTypes` and `disallowedFileTypes` options can be used to limit which file types can be stored or exclude certain file types. The specifications are a list of file extensions and content types separated by semicolons. A type can be specified as a file extension (e.g. `.pdf`) or as a content type (e.g. `application/pdf`).

Example:

```
.txt;application/pdf;.docx
```

Edit localization with keys

(Ticket: EIX-2492)

Two localizations methods are already offered: the translation table and the properties files under the repository path *Administration // Localization // default* or *// custom*.

The translation table has the disadvantage that you cannot assign a neutral key to the localized texts. The key is equal to the value in the first column and its translations must be uniquely assignable.

The disadvantage of the properties files is that you can only dynamically add or change values by checking the file out and in.

The translation table now has a translation key, providing a workaround to these disadvantages. Localized texts can be edited with the well-known `checkinTranslateTerm` function. Users can search for texts related to the translation key with the `findFirstTranslateTerms` function.

Example of how to create a translation term

```
TranslateTerm tterm = new TranslateTerm();
 tterm.setLangs(new String[] {"de", "en"});
 tterm.setTermLangs(new String[] {"Abschließen", "Finish"});
 tterm.setTranslationKey("my.dialog.finish");
 String[] termGuids =
        conn.ix().checkinTranslateTerms(new TranslateTerm[] {tterm}, LockC.NO );
 tterm.setGuid(termGuids[0]);
```

Example of how to find texts in a language

```
Map<String, String> localizedTexts = new HashMap<>();
FindTranslateTermInfo findInfo = new FindTranslateTermInfo();
findInfo.setLangs({"en"});
findInfo.setTranslationKeyPrefix("my.dialog");
FindResult fr = getCon().ix().findFirstTranslateTerms(findInfo, 100);
while (true) {
    Stream.of(fr.getTranslateTerms())
        .forEach(tt -> localizedTexts.put(tt.getTranslationKey(),
                                          tt.getTermLangs()[0]));

    if (!fr.isMoreResults()) break;

    fr = getCon().ix().findNextTranslateTerms(fr.getSearchId(),
                                              localizedTexts.size(),
                                              100);
}
```

Translations can be assigned to a package (`TranslateTerm.getPackageName()`) and a level (`TranslateTerm.getLevel()`). This allows them to be transferred from one repository to another by exporting and importing the package.

Translation keys must be unique across all packages, but only within a level. This enables you to overlap the texts of a key at a higher level.

If there is no level specified in `findFirstTranslateTerms`, the function returns the highest-level texts.

Improvements in handling translations and country and language codes

- Translations in packages can now be exported and imported.
- The *translations* table now supports up to 30 languages.
- *Arabic*, *Danish*, *Greek*, *Finnish*, and *Turkish* have been added to the *translations* table.
- IETF language tags are supported. These can consist of a language code according to ISO 639-1 and optionally be supplemented by a country code according to ISO 3166.
- Language and country are passed to the *ClientInfo* object.

Database search for aspects

Implementation of an API for searching the database for data in aspects.

Keyword lists

Entries in keyword lists can be translated. Keyword lists are no longer bound to an aspect.

SOAP

SOAP is no longer supported.

New file format for import/export

The new format supports metadata 2.0. However, this format is not yet used by default as the client software cannot currently handle this format. If you still want to create a data set in the new format, you can set the option `ix.feature.preview.enable.newgenarchiveexport=true` in the *eloixopt* table.

ELO Textreader (gen. 2)

The ELOix now features a new plug-in that sends documents to ELO Textreader (gen. 2) for full text extraction. The ELO Server Setup makes the necessary settings for this when upgrading.

ES8 file format

The ES8 file format for metadata now includes the JSON representation of an EditInfo object (see ELO Indexserver API documentation). For compatibility reasons, the old format (Windows INI file format) can still be read out for Intray entries. During repository export, ES8 files are still written with the old format.

Example:

```
{
    "docTemplates": null,
    "document": null,
    "keywords": null,
    "markerNames": null,
    "mask": null,
    "maskNames": null,
    "notes": [],
    "pathNames": null,
    "replNames": null,
    "sord": {
    "SReg": "",
    "TStamp": "2022.12.19.15.57.07",
    "acl": "2-7A+PYJA",
    "att": 0,
    "childCount": 1,
    "doc": 0,
    "guid": "(1D82A818-4BF3-40DE-4CBF-147AF0E93FD3)",
    "histCount": 0,
    "id": 749446,
    "info": 0,
    "key": 0,
    "kind": 0,
    "lockId": 0,
    "mask": 1,
    "name": "Ordner123",
    "ownerId": 0,
    "parentId": 1,
    "path": 0,
    "type": 149,
    "vtRep": 0,
    "IDateIso": "20221214095100",
    "XDateIso": "",
    "access": 55,
    "aclItems": [
        {
        "access": 63,
        "id": 9999,
        "name": "Everyone",
        "type": 0,
        "andGroups": null,
        "changedMembers": 0
        }
    ],
    "delDateIso": "",
    "deleted": false,
    "desc": "",
    "details": {
        "archivingMode": 2999,
        "encryptionSet": 0,
        "fulltext": false,
        "sortOrder": 1001,
        "arcReplEnabled": false,
        "fulltextDone": false,
        "replRoot": false,
        "linked": false,
        "incomplete": false,
        "limitedReleaseDocument": false,
        "linkedPermanent": false,
        "documentContainer": false,
        "translateSordName": false,
        "inheritAclDisabled": false,
        "workspace": false,
        "region": false,
        "changedMembers": 0
    },
    "docVersion": null,
    "hiddenText": "",
    "linksComeIn": [],
    "linksGoOut": [],
    "lockName": "Administrator",
    "objKeys": [
        {
        "data": [],
        "displayData": null,
        "id": 0,
        "name": "ELOINDEX",
        "objId": 749446,
        "changedMembers": 15
        }
    ],
    "ownerName": "Administrator",
    "parentIds": [
        "1"
    ],
    "refPaths": null,
    "replNames": [],
    "replSet": {
        "dw": null,
        "dwSync": null,
        "objId": 0,
        "TStamp": "",
        "TStampSync": "",
        "combiGuid": "",
        "changedMembers": 0
    },
    "maskName": "Folder",
    "attVersion": null,
    "deleteDateIso": "",
    "lockIdSord": 0,
    "lockIdDoc": -1,
    "lockNameSord": "Administrator",
    "lockNameDoc": "",
    "TStampSync": "",
    "TStampAcl": "2022.12.14.08.51.46",
    "TStampAclSync": "",
    "deleteUser": -1,
    "aspects": null,
    "spaceGuid": "",
    "spaceGuids": [],
    "TStampLocal": "2022.12.19.15.57.07",
    "packageName": "",
    "regionId": -1,
    "changedMembers": 449242857312616447
    },
    "sordTypes": null,
    "aspectInfos": null,
    "changedMembers": 0
}
```

ELO Textreader (gen. 2)

ELO Textreader (gen. 2) has been completely redeveloped and now starts as a standalone service. It can be invoked from multiple ELO Indexserver instances of different repositories. It allows you to

- Extract text contents
- Create preview files
- Perform OCR on files.

ELO Textreader (gen. 2) replaces the previous ELO Textreader (ELOtr), ELO Preview Converter (ELOpreview), and ELO OCR Service (ELOocr).

For more information, refer to the *ELO Textreader (gen. 2)* documentation:

ELO XML Importer

Extended encryption keys

The concept of "extended encryption keys from ELO 12" has been implemented. The encryption key information from the *config.xml* file is no longer read.

P right

In the control files, the P right can be set for import and and update (`access` tag).

X right

ELO XML Importer supports the X right.

Updating locked documents

Before updating, the system checks whether an ELO document is locked. In this case, the ELO XML Importer waits until an update is possible. No error documents are created.

Receipt files in delete mode

Receipt files are also created in delete mode. This requires the text to be entered in the *config.xml* file to have been configured with the tag `ConfirmDelOk`. The syntax is identical to the syntax of the `ConfirmOk` tag.

ELO Server Setup

The following changes apply to the ELO Server Setup.

Tomcat and JDK

- The setup includes a modified Tomcat 10.0.24.
- Update to Azul OpenJdk 17.0.5

Disable weak TLS MD5 and SHA1 cipher algorithms

The ELO Server Setup 21.1 and higher disables the weak TLS cipher algorithms MD5 and SHA1 for the JVM in the `<elo>\java\conf\security\java.security` configuration file in the `jdk.tls.disabledAlgorithms` parameter.

This prevents client requests with MD5 or SHA1 algorithms when using SSL/TLS encryption in the ELO system.

It also affects HTTPS and TCP connections that use SSL/TLS transport encryption to third-party systems from ELOix or ELOas scripts, ELOix plug-ins, ELO Flows components, and ELO Web Forms apps.

Microsoft SQL Server 2016 secures encrypted database connections with TLS SHA1 algorithms. As a result, the ELO Indexserver cannot establish a database connection to Microsoft SQL Server 2016 when connection encryption is enabled.

To resolve this, you need to turn off connection encryption in Microsoft SQL Server or update SQL Server to at least version 2017.

Encryption of the 'elosetup.conf' file

The setup configuration files `$HOME$\.elosetup.conf` and `<elo>\config\serversetup2\elosetup.conf` are encrypted as of version 21.1.

Accordingly, the ELO Server Setup asks for a password ("main password") in a dialog box every time it is started. This password is used to encrypt and decrypt the configuration files.

Unattended installation

The unattended installation displays a prompt for the main password in the terminal.

To automate this, you can store the password in the environment variable `ELO_SETUP_PASSWORD`.

Decrypt elosetup.conf

A feature in the unattended installation outputs the *elosetup.conf* file as an unencrypted copy.

For more information, refer to the *Installation* documentation, Advanced > Decrypting the setup configuration section.

Automatic certificate renewal

The unattended installation offers a new function for when certificates need to be automatically renewed on a regular basis. This can be the case with ELO systems accessible on the Internet and secured with Let's Encrypt certificates or if certification authorities issue certificates that are valid for less than one year.

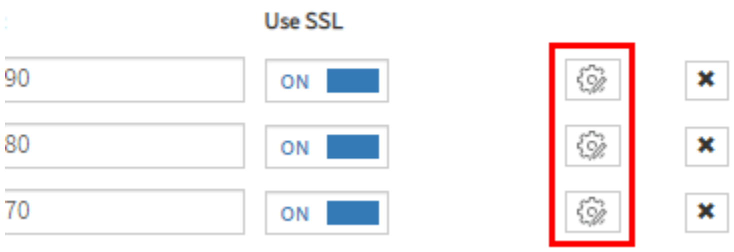
For more information, refer to the *Installation* documentation, Advanced > Unattended installation > Replace SSL certificate section.

Even if certificates are renewed automatically, the configuration files still need to be encrypted. In this case, you need to store the main password in the ELO_SETUP_PASSWORD environment variable referred to in `Unattended installation`.

ELO Textreader (gen. 2)

- ELO Textreader (gen. 2) replaces the *ELO OCR*, *ELO Textreader*, and *ELO Preview Converter* (CV) web applications.
- ELO Textreader (gen. 2) has a configuration file under *<elo>\config\ELO-Textreader-1\config.properties*. All ELO Textreader (gen. 2) settings are made in this file.
- ELO Textreader (gen. 2) is addressed by the ELO Indexservers. You therefore need to enter the URL to ELO Textreader (gen. 2) in the *Indexserver Configure Options* under the parameter *textreaderUrl*.
- The *eloftopt* table is no longer used. For more information, refer to the *ELO Textreader (gen. 2)* documentation (see above).
- The ELO Server Setup removes the Tomcat 4 that the ELO Preview Converter ran on.

Startup options for JVM



- For the Apache Tomcat servers, you can specify additional startup options for the JVM on the *Application Servers* tab via a button (gear icon).
    - Example:

```
-Dhttps.proxyHost=127.0.0.1
-Dhttps.proxyPort=8080
```

Information

Use one line for each setting.

ELO database

The following change was made to the ELO database.

Number of users and groups

The number of users and groups was previously limited to 10,000.

With the `amoptions table, optid 1004`, you were able to increase this limit.

The limit no longer applies as of 21.1 so the option is now obsolete.