ELO COLD

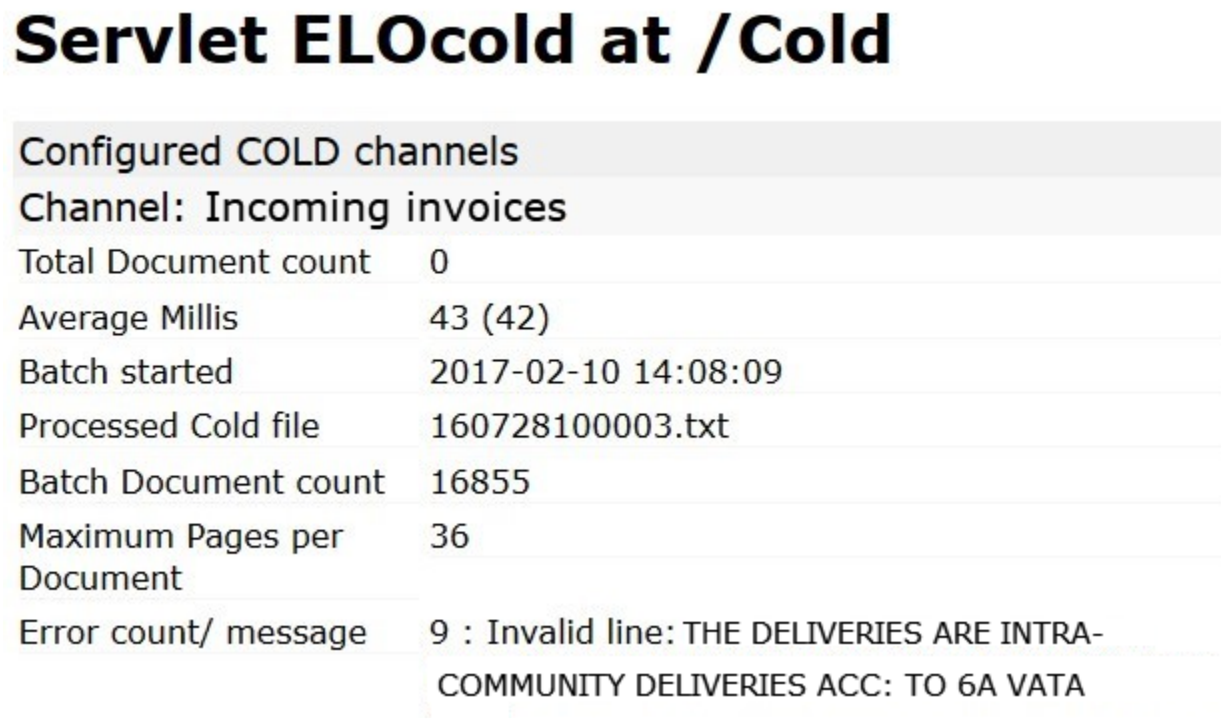
Installation & operation

Table of contents

ELO COLD

Operation

ELO COLD is programmed as a servlet and has a status page that provides information on the current system status.



Servlet ELOcold at /Cold

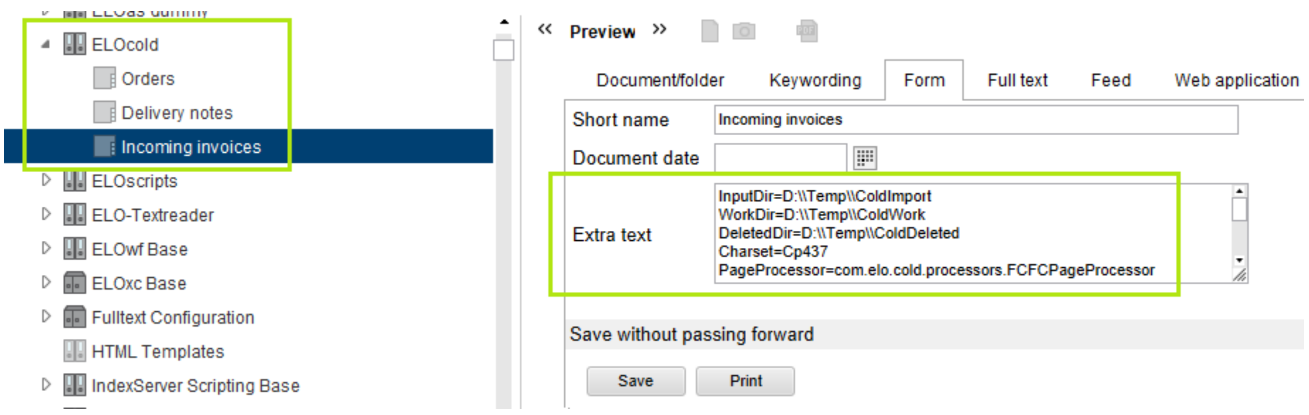
Configured COLD channels	
Channel: Incoming invoices	
Total Document count	0
Average Millis	43 (42)
Batch started	2017-02-10 14:08:09
Processed Cold file	160728100003.txt
Batch Document count	16855
Maximum Pages per Document	36
Error count/ message	9 : Invalid line: THE DELIVERIES ARE INTRA-COMMUNITY DELIVERIES ACC: TO 6A VATA

Configuration and installation

You will have to deploy the ELO COLD WAR archive manually. As with other ELO applications, you have to create a *config.xml* and a *log4j.properties* file and configure them in the Tomcat. The *config.xml* file only contains the parameters required for accessing the ELO Indexserver (IX URL) and the logon data for the COLD user as well as the base path for the configuration. All other configuration data is in the repository. If the base path is blank, */Administration/ELOcold* is applied by default.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<entry key="url">http://srvtdev03:6020/ix-elo100/ix</entry>
<entry key="user">Administrator</entry>
<entry key="password">123</entry>
</properties>
```

ELO COLD can process multiple COLD data streams in parallel. Separate configurations are created for each data stream, which define e.g. the input directories, the index data, or the filing location. This data is filed to the base folder *ELOcold* as a child folder.



Short name: Incoming invoices
Document date:
Extra text:
InputDir=D:\\Temp\\ColdImport
WorkDir=D:\\Temp\\ColdWork
DeletedDir=D:\\Temp\\ColdDeleted
Charset=Cp437
PageProcessor=com.elo.cold.processors.FCFCPageProcessor

On start-up, the system reads all configurations within *ELOcold* and starts a separate processing thread for each of them. The individual converters are read and created from the configuration. The processing pipeline is then started.

The configuration is located in the extra text of the configuration folder.

COLD Pipeline

The following presents and explains a configuration for a COLD pipeline (channel). Custom modules may require additional configuration settings. As the configuration is transferred to the modules completely, it can be easily extended.

```
InputDir=D:\\Temp\\ColdImport
WorkDir=D:\\Temp\\ColdWork
DeletedDir=D:\\Temp\\ColdDeleted
RetryDir=D:\\Temp\\ColdRetry
ErrorDir=D:\\Temp\\ColdError
Charset=Cp437

PageProcessor=com.elo.cold.processors.FCFCPageProcessor
FileIterator=com.elo.cold.processors.FileIteratorBase
LineReader=com.elo.cold.processors.utils.FCFCByteLineReader

Indexer=com.elo.cold.processors.SimpleMatchIndexer
Indexer.MatchLast=my very important customer name at the end of the invoice
Indexer.LineMatcher1=KDNR¶1¶15¶5¶10
Indexer.LineMatcher2=INT.RENR¶1¶*¶RECHNUNGS-NR...:[ ]{1,5}([0-9]+)
Indexer.LineMatcher3=PHONE¶1¶*¶TELEFON:[ ]{1,5}([0-9|\-]{6,15})
Indexer.LineMatcher4=DESTINATION¶1¶*¶RECHNUNGS-NR...:[ ]{1,5}([0-9]{5})
Indexer.LineMatcher5=COMBINED¶Rech={INT.RENR} und Tel={PHONE}
Indexer.Key=RENR¶RE:{INT.RENR}

Renderer=com.elo.cold.processors.RendererBase
Renderer.PageSize=A4
Renderer.TopMargin=30
Renderer.LeftMargin=20
Renderer.FontSize=7
Renderer.LineMargin=9

Archiver=com.elo.cold.processors.ArchiverBase
Archiver.Root=/COLD/{DESTINATION}/2017.01
Archiver.Mask=BARCODE
Archiver.Subject=Cold: {BARC}
```

Global settings are defined in the first section. To keep an overview of what files have already been processed and which haven't in case the program closes, the files are moved to different directories.

Data source

The data source is determined via the *FileIterator* configuration. The standard class for reading directories described below is used as the default setting. However, custom classes can be created and configured, which can use e.g. a database as the data source.

InputDir contains the finished COLD data streams. It may only contain complete, closed files. For this reason, the files should not be created in this directory. Instead, you should create them in a parallel directory and move (and not copy) them to *InputDir* upon completion.

Once the system begins processing a file, it is moved from *InputDir* to *WorkDir*. This ensures that, in case of a file error, the same file isn't imported over and over again, flooding the repository with duplicates. The file remains there during the entire time it is being processed.

After import, the file is moved to *DeletedDir*. This directory can be deleted automatically or moved to a backup. These files are no longer relevant for ELO COLD. If an error occurs during import, the file is moved to *ErrorDir* instead of *DeletedDir.* The administration can decide what to do with the files located here (e.g. correct, file again).

If import is interrupted or cannot be completed due to an error, the file can be submitted for filing again. It is moved to the *RetryDir* directory. If a unique key is entered to the channel definition (*Indexer.Key*), the system can recognize what documents have already been filed and which still have to be processed. If no unique key is entered, this directory is ignored.

Page separation

The LineReader and PageProcessor are responsible for reading lines and separating pages. The standard solution offers two different implementations.

- Normal data streams with line breaks *\r\n* and FormFeed page separation can be read using *com.elo.cold.processors.utils.ByteLineReader* and *com.elo.cold.processors.SimplePageProcessor*.
- Data streams containing control characters in the first column of each line can be processed with *com.elo.cold.processors.utils.FCFCByteLineReader* and *com.elo.cold.processors.FCFCPageProcessor*. Normally, you will have to derive a custom class for interpreting the different control sequences.

The LineReader is responsible for reading files line by line. If necessary, the character set is also converted. From this point on, the system only works with Unicode. The LineReader also indicates whether the end of a page or file has been reached. The end of a file is always interpreted as the implicit end of a page. It is not possible to separate one page over multiple files.

The PageProcessor gathers the read lines until the end of a page is reached. The text page generated is then forwarded to the next pipeline step.

Document separation

The workflow gathers pages from the PageProcessor until the Indexer signalizes a document change. This change can take place on recognition of the first page or on recognition of the last page. Mixed operation is also possible, as long as it remains clear which document pages belong to. To identify the first or last page of a document, the Indexer function *qualifyPage* is called for each page. The Indexer can return whether it has recognized a start or end page. By default, you can search for specific texts indicated via the configuration parameters *Indexer.MatchFirst* and *Indexer.MatchLast*.

Extracting index values

Once all text pages of a document have been read, the document is passed on to the Indexer. You can configure a list of matchers for the Indexer. A matcher is a definition that optionally defines a page and/or line and reads either a fixed workspace or analyzes a regular expression. If there are no page or line limits, you can search all pages or lines.

The definition of a matcher consists of the following parts:

```
Indexer.Matcher<serial number>=<group name>¶<page>¶<line>¶<regular expression>
```

or

```
Indexer.Matcher<serial number>=<group name>¶<page>¶<line>¶<starting column>¶<length>
```

or

```
Indexer.Matcher<serial number>=<group name>¶<composition>
```

The group name is the ELO group from the metadata form definition.

Pages and lines start at 1. If you want to search all pages or lines, you can enter an asterisk * as a placeholder here.

The regular expression can be any RegEx allowed in Java. However, it may only contain one reading area, marked by parentheses. The example `RENR¶1¶*¶INVOICE NO...:[]{1,5}([0-9]+)` defines an expression beginning with `INVOICE NO.…:`. This is followed by 1 to 5 spaces and finished off by a series of numbers (no length restriction). As only the last section is in parentheses, only this part with the numbers is added to the metadata. The rest is simply for identification.

A matcher can compile its contents from other matchers. This compilation consists of fixed texts or the content of other index fields. An index field is integrated by entering the group name in curly brackets. This matcher does not have any side or line information.

With position-dependent indexing, the regular expression is replaced with a starting column and the length. In this case, precisely this area of the line is read. But before saving, leading or trailing spaces are removed via *trim()*. With this variant, a page and a line also have to be entered; searching across areas is not practical here.

As there are comprehensive standard functions available, normally you will not have to create your own Indexer. All you have to do is integrate the SimpleMatchIndexer:

```
Indexer=com.elo.cold.processors.SimpleMatchIndexer
```

Duplicate check

Optionally, the indexer can be assigned a unique key. This is a normal index field whose contents can be compiled from the previously read index fields and fixed text components (as with the short name and filing location).

```
Indexer.Key=RENR¶RE:{INT.RENR}
```

The structure `Indexer.Key=<group name>¶<text or {<group name>}` allows you to use all captured data that can be used to form a unique key. The structure is the same as with the compiled matchers.

The unique key is only checked when the COLD data stream is read from *RetryDir*. In case of an error, a COLD file can be submitted for filing again. All documents that have already been filed are ignored in this repeat run. Only documents the unique key isn't found for are saved.

PDF rendering

As soon as the Indexserver is done, the document is forwarded to the Renderer in the pipeline. The standard Renderer should cover most cases. It is entered to the configuration as follows:

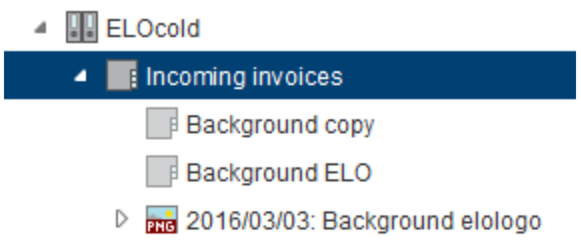
```
Renderer=com.elo.cold.processors.RendererBase
```

The Renderer requires information on the page size, character set used, line height, and margins. This data is provided in the configuration.

In the COLD environment, lines may often be overprinted multiple times e.g. to create an underscore. *Carriage returns* (*\r*) are inserted into a line without *line feed (\n)*. This function is supported by the standard Renderer. More complex functions, such as compound letters (/ + O = Ø) have to be created using a custom implementation. As this identification requires a great deal of work, this loss of time only makes sense if it is actually necessary.

Background images

After rendering text information, background texts or images can also be integrated. These are configured in the COLD channel folder as child folders (for texts) and documents (for pictures).



In the simplest case, you just want to show a text at a specific position. The text, position, text color, and text size are entered to the extra text of the child folder.

```
Message=ELO Digital Office GmbH
Position=1¶20¶-25
FontSize=20
Color=41040
```

The position consists of three parts: the page, the left margin, and the height. You can also enter * as the page; in this case, the text is output on every page. As the coordinates of PDF documents go from bottom to top, it would be inconvenient to show a text at the top edge of the document – you would have to know the page size. However, you can enter a negative value for the height, which is then interpreted as the margin from the top edge.

You can also set the angle of rotation and opacity. The angle is entered in degrees. 0 degrees is the normal text orientation, 90 degrees is 90 degrees counter-clockwise. Opacity is specified in percent. 0 is fully transparent, 100 is fully opaque.

```
Message=Invoice copy
Position=1¶150¶200
FontSize=B80
Color=16744448
Rotation=60
Opacity=25
```

You can enter a B (for *bold*) ahead of the font size.

Image backgrounds only have a position and an opacity setting. Please note that additional image components significantly increase the size of the generated PDF file. Even a small image can easily take up ten times as much space as the entire document text. You should only use this feature sparingly.

```
Position=1¶20¶600
Opacity=10
```

Filing to the repository

The final step is filing to the repository. A standard Archiver is available that should cover most cases.

`Archiver=com.elo.cold.processors.ArchiverBase`

The Archiver identifies the repository target, the short name, and the metadata form and uses this information to create a metadata object for the ELO Indexserver. The PDF file is then created and saved with the document.

The filing target can consist of fixed texts and placeholders, which in turn can consist of hits from the index matcher list. These placeholders are replaced with the group names. The structure is the same as with the unique key or the compiled index fields.

```
Archiver.Root=/COLD/{DESTINATION}/2017.01
```

The same applies for the short name. It can also consist of fixed and dynamic text.

```
Archiver.Subject=Invoice: {RENR}
```

Custom enhancements

Custom analysis modules can easily be created and integrated into the system. They are not subject to any specific naming convention or class hierarchy. All you have to do is implement the defined interface, and the custom class can then be used in the configuration with its full package name. In many cases, it may make sense to derive your custom class from the respective *<*>base.java* class, as you can then use the existing functionality.

Interface	**Implementation**	**Use**
PageProcessorInterface	PageProcessorBase, SimplePageProcessor, FCFCPageProcessor	Read lines with page separation
FileIteratorInterface	FileIteratorBase	List files to be read
LineReaderInterface	ByteLineReader, FCFCByteLineReader	Read text lines and convert character sets
IndexerInterface	IndexerBase, SimpleMatchIndexer	Document separation and index field extraction
RendererInterface	RendererBase	Convert text pages into PDF format
ArchiverInterface	ArchiverBase	Save the PDF document with metadata in the repository

New features

Version 20

As with the other ELO server modules, ELO COLD 20 uses the Logback logger for creating logs. The Logback logger configuration file is named *logback.xml*. The logger settings are entered to this file.

ELO COLD includes the libraries *pdfbox-2.0.18.jar*, *fontbox-2.0.18.jar*, *xmpbox-2.0.18.jar*, and *commons-codec-1.13.jar* as well as the current ELO Indexserver interface *EloixClient.jar*.

Version 21

ELO COLD includes the libraries *pdfbox-2.0.23.jar*, *fontbox-2.0.23.jar*, *xmpbox-2.0.23.jar*, and *commons-codec-1.15.jar* as well as the current ELO Indexserver interface *EloixClient.jar*.