ELO COLD

[Stand: 02.10.2019 | Programmversion: 12.00.002]

Dieses Dokument beschreibt das Modul ELO COLD.

Inhalt

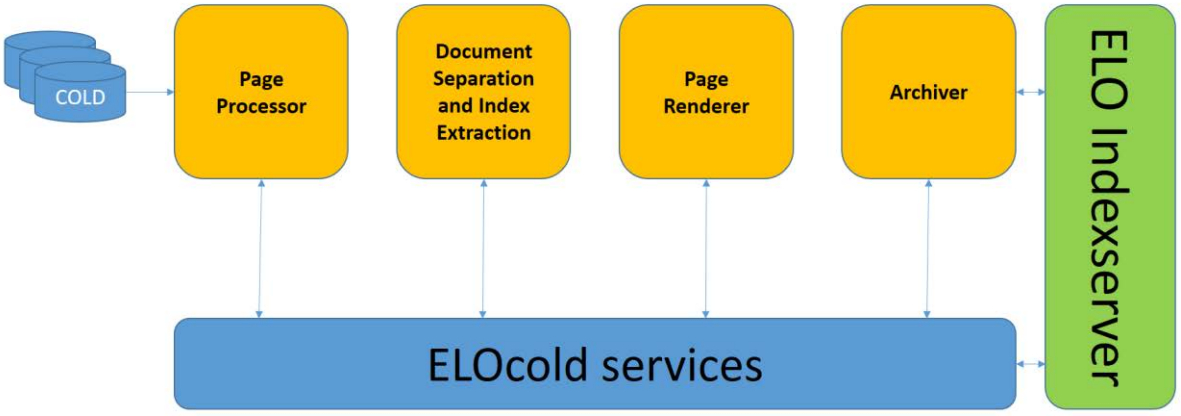
1 Überblick 2
2 Betrieb 3
3 Konfiguration und Installation 4
4 COLD Pipeline 5
4.1 Datenquelle 5
4.2 Seitentrennung 6
4.3 Dokumententrennung 7
4.4 Indexwerte extrahieren 7
4.5 Dublettenkontrolle 8
4.6 PDF-Rendering 8
4.7 Hintergrundbilder 9
4.8 Ablage im Archiv 10
5 Eigene Erweiterungen 11

1 Überblick

Die ELO COLD Services lösen die alte ELOenterprise COLD-Lösung ab, welche in Visual Basic erstellt wurde. Die Vorgehensweise bleibt aber ähnlich: Der COLD-Datenstrom wird programmgesteuert eingelesen, ausgewertet und archiviert. Es gibt keine grafischen Designer und keine proprietäre Konfigurationssprache. Alle Aktionen werden in Java formuliert und stellen somit die volle Funktionalität der Java Umgebung zur Verfügung.

Die Architektur ist als Pipeline angelegt. Aufeinanderfolgende Bausteine sind für das Einlesen der Seiten, die Dokumententrennung, die Gewinnung der Indexdaten, das PDF-Rendering und die Archivablage zuständig. Für jeden dieser Bausteine gibt es ein Interface und eine Basisimplementierung. Wenn im Projekt Anforderungen entstehen, die durch die Basisimplementierung nicht abgedeckt werden, kann man eine eigene Klasse davon ableiten und dort die zusätzlich benötigte Funktionalität programmieren. Für einen Teil der Schritte gibt es bereits im Standard mehrere unterschiedliche Implementierungen.

Die eigenen Implementierungen werden im Classpath der Anwendung abgelegt und werden über die Konfiguration dynamisch geladen. Das gesamte System läuft als Servlet wie jede andere ELO Serveranwendung auch.



2 Betrieb

ELO COLD ist als Servlet programmiert und stellt eine Statusseite zur Verfügung, die über den aktuellen Systemzustand Auskunft gibt.

Servlet ELOcold at /Cold

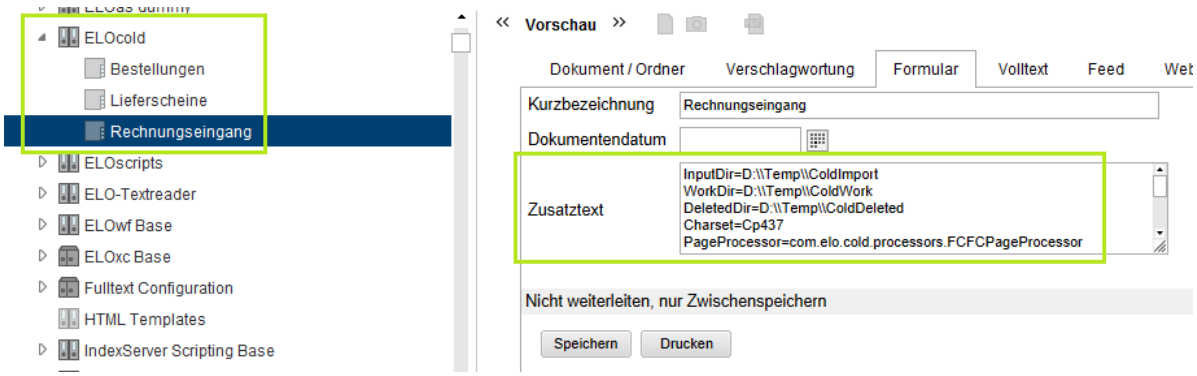
Configured COLD channels	
Channel: Rechnungseingang	
Total Document count	0
Average Millis	43 (42)
Batch started	2017-02-10 14:08:09
Processed Cold file	160728100003.txt
Batch Document count	16855
Maximum Pages per Document	36
Error count/ message	9 : Invalid line: DIE LIEFERUNGEN SIND ALS INNERGEMEINSCHAFTLICHE LIEFERUNGEN LT. Σ 6A U

3 Konfiguration und Installation

Das WAR-Archiv von ELO COLD muss man manuell deployen. Wie bei anderen ELO Anwendungen muss eine config.xml und eine log4j.properties Datei angelegt werden und im Tomcat konfiguriert werden. Die config.xml Datei enthält nur die notwendigen Parameter zum Zugriff auf den Indexserver (IX-URL) und die Anmeldedaten für den COLD-User sowie den Basispfad für die Konfiguration. Alle weiteren Konfigurationsdaten liegen im Archiv. Wenn der Basispfad fehlt, wird er automatisch mit „/Administration/ELOcold“ angenommen.

```
<?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">elo</entry>
</properties>
```

ELO COLD kann mehrere COLD-Datenströme parallel verarbeiten. Zu jedem Datenstrom wird eine eigene Konfiguration hinterlegt, welche z. B. die Eingangsverzeichnisse, die Indexdaten oder den Ablageort definieren. Diese Daten werden als Unterordner im Basisordner *ELOcold* abgelegt.



Beim Start liest das System alle Konfigurationen unterhalb von *ELOcold* ein und startet für jede einen eigenen Verarbeitungs-Thread. Aus der Konfiguration werden dann die einzelnen Konverter ausgelesen und erzeugt. Anschließend wird die Verarbeitungs-Pipeline gestartet.

Die Konfiguration liegt jeweils im Zusatztext des Konfigurationsordners.

4 COLD Pipeline

Im Folgenden wird eine Konfiguration für eine COLD Pipeline (Kanal) aufgezeigt und erläutert. Dabei können in eigenen Modulen zusätzliche Konfigurationseinstellungen notwendig werden. Da die Konfiguration jeweils komplett an alle Module übergeben wird, kann diese leicht erweitert werden.

```
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}
```

Im ersten Abschnitt werden globale Einstellungen definiert. Damit beim Programmabbruch deutlich wird, welche Dateien bereits verarbeitet wurden und welche noch nicht, werden die Dateien während des Durchlaufs in unterschiedliche Verzeichnisse verschoben.

4.1 Datenquelle

Die Datenquelle wird über die Konfiguration *FileIterator* bestimmt. Als Voreinstellung wird die unten beschriebene Standardklasse für das Einlesen von Verzeichnissen verwendet. Es können aber eigene Klassen hinterlegt und konfiguriert werden, die als Datenquelle z. B. eine Datenbank verwenden können.

Das *InputDir* enthält die fertigen COLD-Datenströme. Es darf nur vollständige und abgeschlossene Dateien beinhalten. Aus diesem Grund sollten die Dateien nicht in diesem Verzeichnis erstellt werden. Stattdessen sollten sie in einem parallelen Verzeichnis erzeugt werden und nach Fertigstellung in das *InputDir* verschoben werden (nicht kopiert).

Sobald das System mit der Verarbeitung einer Datei beginnt, wird diese aus dem *InputDir* in das *WorkDir* verschoben. Somit ist sichergestellt, dass im Falle eines Dateifehlers nicht die gleiche Datei immer und immer wieder importiert wird und das Archiv mit Duplikaten überschwemmt. Dort bleibt die Datei auch während der kompletten Bearbeitungszeit liegen.

Nach dem Abschluss des Imports wird die Datei in das *DeletedDir* verschoben. Dieses Verzeichnis kann durch einen automatisierten Vorgang gelöscht oder in ein Backup verschoben werden. Für ELO COLD sind diese Dateien nicht mehr relevant. Falls während des Imports ein Fehler aufgetreten ist, wird die Datei nicht in das *DeletedDir*, sondern in das *ErrorDir* verschoben. Hier kann der Administrator dann entscheiden, wie mit diesen Dateien weiter zu verfahren ist (Korrigieren, erneut Ablegen…).

Falls ein Import-Lauf unterbrochen wurde oder wegen eines Fehlers nicht vollständig durchlaufen wurde, kann die Datei erneut zur Ablage vorgelegt werden. Hierzu wird sie in das Verzeichnis *RetryDir* verschoben. Wenn in der Kanaldefinition ein eindeutiger Schlüssel hinterlegt ist (Indexer.Key), kann das System erkennen, welche Dokumente bereits abgelegt wurden und welche noch bearbeitet werden müssen. Wenn es keinen eindeutigen Schlüssel gibt, wird dieses Verzeichnis ignoriert.

4.2 Seitentrennung

Für das zeilenweise Einlesen und die Seitentrennung sind der LineReader und der PageProcessor zuständig. Im Standard gibt es zwei unterschiedliche Implementierungen.

- Normale Datenströme mit Zeilenumbrüchen \r\n und FormFeed-Seitentrennungen können über das Modul *com.elo.cold.processors.utils.ByteLineReader* sowie *com.elo.cold.processors.SimplePageProcessor* eingelesen werden.
- Datenströme, welche Steuerzeichen in der ersten Spalte jeder Zeile beinhalten, können mittels *com.elo.cold.processors.utils.FCFCByteLineReader* und *com.elo.cold.processors.FCFCPageProcessor* verarbeitet werden. Hier wird man allerdings im Normalfall eine eigene Klasse zur Interpretation der jeweils unterschiedlichen Steuersequenzen ableiten müssen.

Der LineReader ist für das zeilenweise Einlesen der Datei zuständig. Hier wird auch eine eventuell notwendige Zeichensatzkonvertierung durchgeführt. Ab dieser Stelle wird intern nur noch mit Unicode gearbeitet. Er teilt auch mit, ob ein Seitenende oder das Dateiende erreicht wurden. Ein Dateiende wird immer als implizites Seitenende interpretiert. Es ist also nicht möglich, eine Seite über mehrere Dateien zu verteilen.

Der PageProcessor sammelt die eingelesenen Zeilen nacheinander auf, bis ein Seitenende erreicht wird. Die daraus generierte Textseite wird dann an den nächsten Pipeline-Schritt weitergeleitet.

4.3 Dokumententrennung

Die Ablaufsteuerung sammelt die Seiten aus dem PageProcessor so lange, bis der Indexer einen Dokumentenwechsel signalisiert. Der Wechsel kann sowohl durch Erkennung der ersten Seite wie auch durch Erkennung der letzten Seite erfolgen. Auch ein Mischbetrieb ist möglich, solange die Dokumentenzuordnung eindeutig bleibt. Zur Erkennung der ersten oder der letzten Seite eines Dokuments wird zu jeder Seite die Indexer-Funktion *qualifyPage* aufgerufen. Hier kann der Indexer zurückgeben, ob er eine Start- oder Endeseite erkannt hat. Im Standard kann man hier nach bestimmten Textteilen suchen, die über die Konfigurationsparameter *Indexer.MatchFirst* sowie *Indexer.MatchLast* angegeben werden.

4.4 Indexwerte extrahieren

Sobald alle Textseiten eines Dokuments eingelesen sind, wird das Dokument an den Indexer weitergegeben. Zum Indexer kann man eine Liste von Matchern konfigurieren. Ein Matcher ist eine Definition, welche optional eine Seite und/oder eine Zeile festlegt und dort entweder einen festen Zeichenbereich ausliest oder einen regulären Ausdruck auswertet. Falls keine Seiten oder Zeileneinschränkung vorgenommen werden, kann man auch alle Seiten oder alle Zeilen durchsuchen lassen.

Die Definition eines Matchers besteht aus den folgenden Teilen:

```
Indexer.Matcher<Laufende Nummer>=<Gruppenname>¶<Seite>¶<Zeile>¶<regulärer
Ausdruck>
```

oder

```
Indexer.Matcher<Laufende
Nummer>=<Gruppenname>¶<Seite>¶<Zeile>¶<Startspalte>¶<Länge>
```

oder

```
Indexer.Matcher<Laufende Nummer>=< Gruppenname>¶<Zusammensetzung>
```

Der **Gruppenname** ist die ELO Gruppe aus der Maskendefinition.

Seiten und **Zeilen** werden von 1 an aufwärts gezählt. Falls alle Seiten oder alle Zeilen durchsucht werden sollen, kann hier auch ein Stern (*) als Platzhalter eingesetzt werden.

Der **reguläre Ausdruck** kann jeder in Java zulässige RegEx-Ausdruck sein. Er darf aber nur eine Auslesezone enthalten, die durch runde Klammern markiert wird. Im Beispiel *RENR¶1¶*¶RECHNUNGS- NR...:[ ]{1,5}([0-9]+)* wird ein Ausdruck definiert, der mit dem Text „RECHNUNGS-NR…:“ beginnt. Danach folgen 1 bis 5 Leerzeichen. Zum Abschluss kommt dann eine Folge von Ziffern (ohne Längeneinschränkung). Da nur dieser letzte Abschnitt in den runden Klammern steht, wird auch nur dieser Teil (die Ziffern) in die Verschlagwortung übernommen. Der Rest dient nur zur Erkennung.

Ein Matcher kann seinen Inhalt auch aus anderen Matchern zusammensetzen. Die **Zusammensetzung** besteht aus festen Texten oder dem Inhalt anderer Indexzeilen. Eine Indexzeile wird eingebunden, indem der Gruppenname in geschweiften Klammern aufgeführt wird. So ein Matcher hat natürlich keine Seiten- oder Zeileninformation.

Bei einer positionsabhängigen Indizierung folgen statt des regulären Ausdrucks eine **Startspalte** und eine **Längen**angabe. In diesem Fall wird genau dieser Bereich der Zeile ausgelesen. Vor der Speicherung werden allerdings führende oder nachfolgende Leerzeichen mittels *trim()* entfernt. Bei dieser Variante muss auch eine Seite und eine Zeile angegeben werden, eine Suche über Bereiche ist hier nicht sinnvoll anwendbar.

Da die Fähigkeiten im Standard schon recht umfangreich sind, wird man im Normalfall keinen eigenen Indexer erstellen müssen. Es muss lediglich der SimpleMatchIndexer eingebunden werden:

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

4.5 Dublettenkontrolle

Der Indexer kann optional mit einem eindeutigen Schlüssel ausgerüstet werden. Dabei handelt es sich um eine normale Indexzeile deren Inhalt aus den zuvor erfassten Indexzeilen und festen Textbestandteilen zusammengesetzt werden kann (analog zur Kurzbezeichnung und zum Ablageziel).

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

Der Aufbau `Indexer.Key=<Gruppenname>¶<Text oder {<Gruppenname>}` erlaubt die Verwendung aller erfassten Daten, aus denen dann ein eindeutiger Schlüssel gebildet werden kann. Der Aufbau ist der gleiche wie bei den zusammengesetzten Matchern.

Der eindeutige Key wird nur geprüft, wenn der COLD-Datenstrom aus dem *RetryDir* gelesen wird. Somit ist es möglich, dass im Fehlerfall eine COLD-Datei erneut zu Ablage vorgelegt werden kann. Alle Dokumente, welche bereits archiviert wurden, werden bei diesem erneuten Durchlauf ignoriert. Nur Dokumente, zu denen der eindeutige Schlüssel nicht gefunden wird, werden gespeichert.

4.6 PDF-Rendering

Sobald der Indexer seine Arbeit abgeschlossen hat, wird das Dokument in der Pipeline an den Renderer weitergereicht. Auch hier wird man in vielen Fällen mit dem Standard-Renderer zurechtkommen. Er wird so über die Konfiguration eingetragen:

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

Der Renderer benötigt einige Angaben zur Seitengröße, verwendetem Zeichensatz, Zeilenhöhe und Rändern. Diese Daten werden über die Konfiguration übergeben.

Im COLD-Umfeld kommt es häufiger vor, dass Zeilen mehrfach überdruckt werden, z. B. um einen Unterstrich zu erzeugen. Hierzu werden in eine Zeile Wagenrückläufe (Carriage Return, \r) ohne Zeilenvorschub (Line Feed, \n) eingefügt. Diese Funktion wird bereits vom Standard-Renderer unterstützt. Komplexere Funktionen, wie z. B. zusammengesetzte Buchstaben (/ und O zu Ø) müssen durch eine eigene Implementierung erzeugt werden. Da diese Erkennung einen erheblichen Mehraufwand erzeugt, ist dieser Zeitverlust nur sinnvoll, wenn er tatsächlich notwendig ist.

4.7 Hintergrundbilder

Nach dem Rendern der Textinformation können zusätzlich Hintergrundtexte oder Bilder eingebunden werden. Diese werden im Ordner des COLD-Kanals als Unterordner (für Texte) und Dokumente (für Bilder) hinterlegt.



Im einfachsten Fall will man nur einen Text an einer bestimmten Stelle einblenden. Hierzu wird im Zusatztext des Unterordners der auszugebende Text, die Position und die Textfarbe und Größe hinterlegt.

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

Die Position besteht aus drei Teilen: der Seite, dem linken Rand und der Höhe. Als Seitenangabe kann man auch ein * angeben. In diesem Fall wird der Text auf jeder Seite ausgegeben. Da die Koordinaten von PDF-Dokumenten von unten nach oben laufen, wäre es umständlich, wenn man einen Text am oberen Rand des Dokuments anzeigen möchte – man müsste hierzu die Seitengröße kennen. Dafür gibt es die Möglichkeit, dass man einen negativen Wert für die Höhe einträgt. Dieser wird dann als Abstand vom oberen Rand gewertet.

Weiterhin kann man einen Drehwinkel und die Transparenz einstellen. Der Winkel wird in Grad eingegeben. 0 Grad ist die normale Textrichtung, 90 Grad ist senkrecht nach oben (also gegen den Uhrzeigersinn). Die Transparenz (oder genauer die „Undurchsichtigkeit“) wird in Prozent angegeben. 0 ist völlig durchsichtig, 100 ist völlig undurchsichtig.

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

Bei der Textgröße kann ein B (für Bold) für den Fettdruck vorangestellt werden.

Bildhintergründe haben nur eine Position und eine Transparenzeinstellung. Man muss beachten, dass zusätzliche Bildanteile das Volumen der erzeugten PDF-Datei massiv vergrößern. Selbst ein kleines Bild kann leicht 10-mal mehr Speicher belegen als der komplette Dokumenttext. Aus diesem Grund sollte man dieses Feature nur sehr sparsam einsetzen.

```
Position=1¶20¶600
Opacity=10
```

4.8 Ablage im Archiv

Am Ende der Pipeline steht dann die Archivablage. Auch hier steht ein Standard-Archiver zur Verfügung, der in vielen Fällen ausreichen sollte.

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

Der Archiver ermittelt das Archivziel, die Kurzbezeichnung und die Verschlagwortungsmaske, und erzeugt daraus ein Verschlagwortungsobjekt für den Indexserver. Anschließend wird die PDF-Datei erstellt und zum Dokument abgespeichert.

Das Archivziel kann aus festen Texten und aus Platzhaltern bestehen, die aus Treffern der Index-Matcher- Liste bestehen können. Diese Platzhalter werden über den Gruppennamen aufgelöst. Der Aufbau ist der gleiche wie bei dem eindeutigen Schlüssel oder den zusammengesetzten Indexzeilen.

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

Das gleiche gilt für die Kurzbezeichnung. Sie kann ebenfalls aus festen und dynamischen Textteilen zusammengesetzt werden.

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

5 Eigene Erweiterungen

Eigene Auswertungsmodule können leicht erstellt und in das System integriert werden. Sie unterliegen keinen speziellen Namenskonventionen oder einer bestimmten Klassenhierarchie. Es muss lediglich das definierte Interface implementiert werden und anschließend kann die eigene Klasse mit vollem Paketnamen in der Konfiguration verwendet werden. In vielen Fällen dürfte es aber sinnvoll sein, die eigene Klasse von der jeweiligen *<*>base.java* Klasse abzuleiten, da man dann die bereits vorhandene Funktionalität mitverwenden kann.

Interface	Implementierung	Verwendung
PageProcessorInterface	PageProcessorBase SimplePageProcessor FCFCPageProcessor	Einlesen der Zeilen mit Seitentrennung
FileIteratorInterface	FileIteratorBase	Auflisten der einzulesenden Dateien
LineReaderInterface	ByteLineReader FCFCByteLineReader	Einlesen der Textzeilen und Zeichensatzumwandlung
IndexerInterface	IndexerBase SimpleMatchIndexer	Dokumententrennung und Extraktion der Indexmerkmale
RendererInterface	RendererBase	Konvertierung der Textseiten in ein PDF-Format
ArchiverInterface	ArchiverBase	Speichern des PDF-Dokuments mit Verschlagwortung im Archiv