Matthias Thiele
ELO Digital Office GmbH
Tübinger Straße 43
70178 Stuttgart
www.elo.com



ELO SCRIPT

99 SKRIPTBEISPIELE FÜR ELO

Skriptbeispiele für ELOprofessional und ELOenterprise aus den Bereichen Ablage, Dokumente, Verschlagwortung und Workflow.

INHALTSVERZEICHNIS

Inhalt

Vorwort 1

Ziel dieses Buchs 1

An wen richtet es sich 1

Wie geht es weiter 1

Warum sind es nicht genau 99 Beispiele 2

Warum gibt es die Beispiele nicht im Download 2

[10] Zweite Auflage 2

[10.1] Dritte Auflage 2

[10.2] Vierte Auflage 3

[11.0] Fünfte Auflage 3

[12.0] Sechste Auflage 3

Workflow 4

1-Click Workflow-Start 4

Automatische Benachrichtigung bei Workflowaufgaben 5

Mail aus einem Workflow heraus versenden 10

Workflow im Java Client weiterleiten 12

Workflowaufgabe per Mail ankündigen 13

Workflowstatus per Skript ändern 15

Workflowknoten entziehen 15

Workflow-Weiterleiten Action Buttons 16

[10] Skript-Event beim Aktualisieren der Aufgabenliste 18

[10] Workflow-Diagramm per Skript bearbeiten 18

[10] Parent-Workflow eines Sub-Workflows ermitteln 18

[10.1] Workflow-Annahme verhindern (Indexserver) 19

Java Client allgemein 21

Skripte in mehreren Sprachen 21

Externe JAR Dateien in Skripte einbinden 21

Externe Programme aufrufen 22

Haftnotizen in der Postbox einfügen 23

CommandLink Dialog 24

CommandLink Dialog aus einer Ordnerauswahl 25

INHALTSVERZEICHNIS

Annotations verlinken 27
Verwendung des http Automation Interface 30
Nachrichten zwischen unterschiedlichen Fenstern versenden 32
Skript über die COM Schnittstelle aufrufen 32
[10.2] Verwendung der OLE Automation Emulation 34
Datei in der Postbox kopieren 34
Versionsabhängige Skriptfunktionen 35
Externe Viewer einbinden 35
Dateiauswahl-Dialog 36
EML oder MSG Mails auswerten 37
Intervallgesteuerte Aktionen ausführen 37
Hintergrundprozess starten 38
Hintergrundprozess vorzeitig beenden 40
Prozess mit Protokolleinträgen 41
Hintergrundprozess mit Fortschrittsanzeige 42
Konfiguration für Skripte 45
HTML Text ins Windows Clipboard einfügen 46
URL Encoding ermitteln 47
[10] Zweite Seite bei der Anzeige aufblättern 48
[10] Haftnotiz im Dokument per Skript erzeugen 49
[10] Anzeige der Wang Annotations 50
[10] Aktenzeichen generieren 51
[10.1] Alle EXIF Tags einer Tiff/Jpeg Datei anzeigen 52
[10.1] Workflow-Annahme verhindern (Java Client) 54
[10.1] Barcode – Handscanner 54
[10.1] Automatisches CheckIn über die Skript Schnittstelle 56
[10.1] Zoom-Faktor per Skript einstellen 56
[10.1] Beispiel für einen einfachen Datei-Import 57
[10.2] Versionsnummern per Skript generieren 59
[10.2] PDF Viewer per Skript umschalten 60
[10.2] Dateinamen für Drag&Drop Operation verändern 60
[10.2] Checkout Verhalten ändern 61

INHALTSVERZEICHNIS

[11.0] Prüfen, ob man Mitglied einer Gruppe ist 62
[11.0] http-Get Aufruf in einen String laden 62
[12.0] Noch nicht gespeicherte Eingaben melden 63
[12.0] Event bei der Selektionsänderung 63
[12.0] Event am Ende eines Kopiervorgangs 64
[12.0] Selektierten Archiveintrag ermitteln 64
[12.0] Skript-Events im Mobile Connector 65
Dropzone und Java Client 67
Skriptkachel mit Ok/ Fehleranimation 67
Trim auf einen gefundenen Ausdruck anwenden 67
Mailversand über die Dropzone 68
Title-Erkennung bei mehrdeutigen Einträgen 70
Automatische Ablage aus dem PDF Drucker 72
Tabellen in regulären Ausdrücken 72
[10] Dropzone Import über eine URL 73
[10.2] Bereichssuche über das aktuelle Jahr oder den aktuellen Monat 75
[11.0] Aufruf einer Java Client Kachel über die Dropzone 75
[11.0] Favoriten-Suche über die Dropzone 76
[11.0] Angelegte Ordnerstruktur bearbeiten 77
[12.0] Rückmeldung einer Skriptkachel 78
Archivinformationen 79
Archivpfade eines Eintrags ermitteln 79
Einen bestimmen Archivpfad aufblättern 79
Nur leere Ordner löschen 80
Eine bestimmte Seite im Dokument aufblättern 81
Liste der angemeldeten Anwender ermitteln 81
Prüfen, ob ein Dokument vorhanden ist 81
Custom-Reporteintrag erzeugen 82
Benutzereinstellungen speichern 82
Externe Dateien ins Archiv aufnehmen 83
[10] Anzeigelisten nach dem Sammeln bearbeiten 85
[10] Prüfen, ob eine Datei im Archiv vorhanden ist 86

INHALTSVERZEICHNIS

[10] Anwenderordner „Persönlicher Bereich“ finden 88
[10] Dublettenprüfung im Archiv 88
[10.2] Volltext-Information erneuern 88
Toolbar und Navigation im Java Client 92
Intranet Seite im Java Client einblenden 92
Button mit Tooltip erstellen 96
[10] Verschlagwortungsabhängige Anzeige 96
[10] Namen der Ansichten 98
[10] Suchbegriff automatisch in die Trefferanzeige übernehmen 98
[10.1] Funktionsaufruf per Hotkey (Click&Find) 99
[11.0] Workflow-Aufgabe anzeigen 100
Verschlagwortung im Java Client 101
Eingabefeld beschreibbar machen 101
Automatisches Zählerfeld 101
Button hinter einer Indexzeile anzeigen 102
Tab Ausblenden (1) 103
Tab ausblenden (2) 104
Automatisches CheckIn beim Speichern 104
Wie wurde der Verschlagwortungsdialog geschlossen 105
Bild im Verschlagwortungsdialog anzeigen 105
Frei definierbare Stichwortlisten 107
String an das Windows Clipboard übergeben 108
Adresse in Google Maps anzeigen 109
[10] Trefferliste bearbeiten 111
[10] OCR Click-Wort in Skriptdialogfeld übernehmen 111
[10] Mails mit Attachment markieren 114
[10] Nicht-modaler Verschlagwortungsdialog 114
[10.2] Verschlagwortung kopieren 115
[10.2] Verschlagwortungsmaske durch Anwenderauswahl bestimmen 117
Microsoft Office Einbindung 119
Formularfelder beimCheckOut füllen 119
Formularfelder aus einem Skript lesen oder schreiben 121

INHALTSVERZEICHNIS

Automatischer CheckIn beim Schließen von Word 129
Neues Dokument aus externer Quelle erzeugen 130
Excel Dokument aus Verschlagwortung füllen 131
Hängende Excel Versionen 134
ELO Automation Services 136
Microsoft Office Dokument in PDF umwandeln 136
Feed Beitrag aus ELOas erstellen 136
Indexinformation aus dem Volltext gewinnen 136
Rückgabewerte aus einer direkt Rule 137
Workflow an mehrere Nachfolger weiterleiten 137
Nachfolgerliste dynamisch erweitern 138
Anwender eines Workflowknotens ändern 140
Datumsfeld ein Jahr weiterzählen 140
Nach mehreren Indexbegriffen suchen 141
Eigene Suche ausführen 141
Fremde Web Seiteninhalte auslesen 141
Lesen und Schreiben von Map Feldern 142
Verschlüsselte Passwörter entschlüsseln 143
ELO Dokumententyp aus der Datei-Extension ermitteln 143
Dokument auf einen anderen Dateipfad umziehen 144
Suche nach Zeitstempel 144
Wasserzeichen in PDF Dokument einfügen 145
Bearbeitungsreihenfolge im TreeWalk 146
Maskenübergreifende Rulesets 147
Aktive Workflow-Termine zu gelöschten Dokumenten 147
[10] Bearbeitungsliste aus einer registrierten Indexserver-Funktion 148
[10] Alle Dokumententypen beim TreeWalk bearbeiten 149
[10.1] Tiff Dokumente klammern 150
JavaScript allgemein 151
Erstes Array Element löschen 151
Property oder Funktionsschreibweise 151
Zeilenumbrüche in regulären Ausdrücken 151

INHALTSVERZEICHNIS

[10.1] Switch – case mit Java String Objekten ... 152
[10.1] Dateizugriff aus Skripten ... 152
[12.0] Datumsformatierung ... 154
Indexserver-Funktionen ... 156
Anmeldung kontrollieren ... 156
Indexzeilenwerte beim Weiterleiten füllen (1) ... 156
Indexzeilenwerte beim Weiterleiten füllen (2) ... 158
Konsistenzprüfungen beim Speichern ... 158
Annotations beim CheckIn löschen ... 160
Previewerstellung für Dokumente ... 161
[10.1] Indexserverskripte mit erhöhten Berechtigungen ausführen ... 161
[10.2] Blacklist für bestimmte Dokumententypen ... 162
[10.2] Externe Dateien im Archiv anzeigen ... 163
Formulare ... 166
Nachricht an den Client senden ... 166
Nachricht an ein Formular senden ... 167
Spaltenbreite in einem Formular setzen ... 169
Umfangreiche Skripte in Formularen ... 169
Checkbox als Stempel verwenden ... 170
Name aus Array-Variablen ermitteln ... 172
Index aus Array-Variablen ermitteln ... 172
Zugriff auf den Indexserver aus einem Formular ... 173
Filter für Feldinhalte ... 173
Tab sperren ... 175
[10] Eigene Bilder im Formular anzeigen ... 176
[10.1] Eigene Hintergrundbilder im Formular ... 178
[11.0] Formularfeld in Abhängigkeit eines Wertes einfärben ... 179
Indexserver Events ... 181
Workflowweiterleitung erzeugt Randnotiz ... 181
Mail für neue Workflowtermine ... 182
WebApps ... 185
Listen in ELO Web Apps ... 185

INHALTSVERZEICHNIS

Indexserveraufrufe über die JSON API 185
Konfigurationsvariablen im Scope einer App 187
Drag & Drop 188

Vorwort

ZIEL DIESES BUCHS

Dieses Buch stellt eine Sammlung verschiedener Skriptbeispiele für Automatisierungen im ELOprofessional und ELOenterprise Umfeld (in Zukunft nur noch als ELO bezeichnet) dar. Die Beispiele sind vorwiegend intern entstanden und wurden zum Teil bereits im Forum veröffentlicht. Ein Teil der Beispiele wurden aber auch neu extra für dieses Werk erstellt.

Ziel ist es, dem ELO Consultant, der eine bestimmte Aufgabe beim Kunden lösen muss, einen Pool von Ideen und Vorschlägen als Hilfestellung an die Hand zu geben. In vielen Fällen wird das Beispiel nicht 100% der Kundenanforderungen abdecken. Aber es ist zumindest ein Startpunkt, der dann passend erweitert werden muss.

Manchmal sieht man in der Dokumentation eine Schnittstelle, deren Befehle zwar ordentlich beschrieben sind, die Verwendung insgesamt aber unklar bleibt. Auch hier soll diese Sammlung weiterhelfen und komplette Verwendungsbeispiele statt nur einzelner Funktionsbeschreibungen bieten.

Die einzelnen Beispiele sind in verschiedene Kategorien eingeteilt. Gerade bei etwas umfangreicheren Skripten ist die Zuordnung manchmal schwierig, da unterschiedliche Aufgaben erledigt werden. Deshalb lohnt es sich, auch in verwandten Bereichen nach Beispielen zu suchen, wenn man ein bestimmtes Problem lösen möchte.

AN WEN RICHTET ES SICH

Dieses Buch setzt Kenntnisse der Skriptprogrammierung in JavaScript voraus. Ein erfahrener VB Skript Entwickler wird viele Beispiele verstehen – trotzdem ist es sinnvoll, sich vorab mit der Sprache auseinander zu setzen bevor man in die ELO Schnittstellen eintaucht. Das Buch eignet sich nicht zum Erlernen der Skriptprogrammierung da die Beispiele nicht didaktisch aufbereitete sind, nicht aufeinander aufbauen und die Einzelfunktionen im Normalfall nicht erklärt werden.

WIE GEHT ES WEITER

Dieses Buch möchte die breite Verwendung der verschiedenen Skriptschnittstellen im ELO Umfeld darstellen. Das führt aber dazu, dass nicht an jeder Stelle in die Tiefe gegangen werden kann. Hier helfen dann die JavaDoc API Beschreibungen des Indexservers, Java Clients oder ELOas weiter.

Insbesondere die Indexserverschnittstelle ist das A und O im ELO Umfeld. Egal, auf welcher Ebene man arbeitet – Java Client, ELOas, Web Client, Workflow Formulare, ELO Web Apps – über kurz oder lang wird man auf die Indexserver Schnittstelle und die Objekte dieser Schnittstelle treffen. Eine gute Kenntnis dieser API hilft also bei allen Programmieraufgaben weiter.

VORWORT

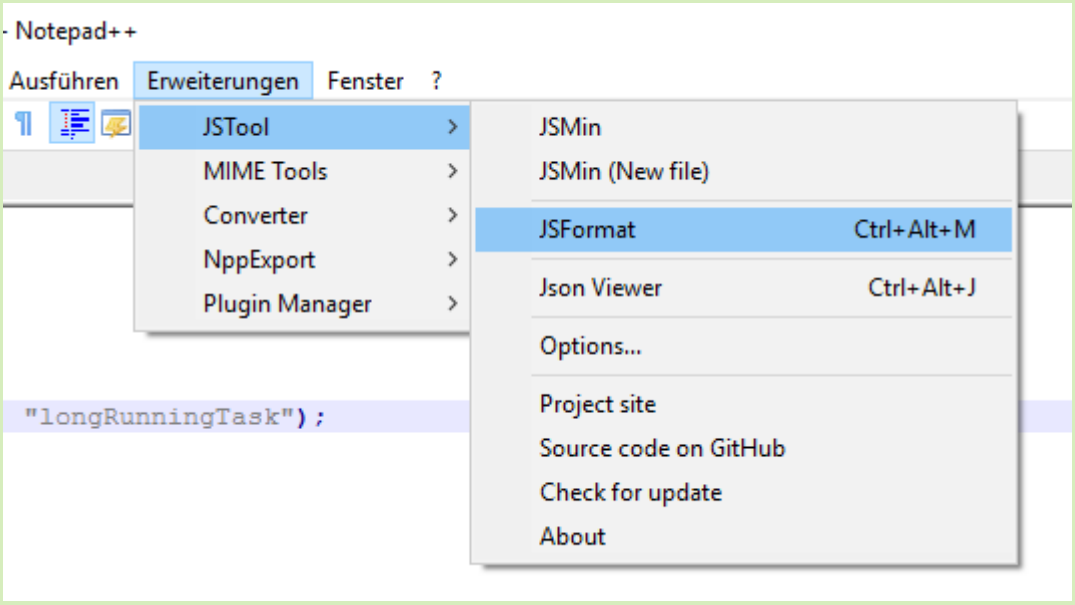
WARUM SIND ES NICHT GENAU 99 BEISPIELE

Der Titel ist ein Arbeitstitel, der relativ schnell veralten wird – aus der Praxis werden ständig neue Beispiele kommen und das Dokument wird entsprechend erweitert werdem. Zudem: 99 ist eine schön griffige Zahl für einen Titel.

WARUM GIBT ES DIE BEISPIELE NICHT IM DOWNLOAD

Dieses Buch liefert keine fertigen Musterlösungen. Es soll nur erklären, wie etwas prinzipiell erledigt werden kann. Die Beispiele sind deshalb kurzgehalten und beschränken sich oft auf eine Teilfunktion. Zum Teil sind sie ohne weitere Rahmenprogrammierung gar nicht ausführbar.

Wenn man die Beispiele per Copy&Paste aus dem PDF Dokument herauskopiert, geht oft die Formatierung kaputt. Hierzu gibt es Tools, die automatisch eine sinnvolle Formatierung erzeugen, für Notepad++ z.B. das PlugIn JSTool. Es gibt aber auch stand alone Tools, die so etwas machen.



[10] ZWEITE AUFLAGE

Zur Version 10 kommt diese überarbeitete Version mit zusätzlichen Beispielen für ELOprofessional/ ELOenterprise 10. Damit man diese Erweiterungen leichter erkennt, wurden sie in der Überschrift durch das Präfix [10] gekennzeichnet.

Die neuen Beispiele sollten zusätzliche Funktionen und ihre Verwendung aufzeigen. In älteren ELO Versionen werde diese im Normalfall nicht funktionsfähig sein.

[10.1] Dritte Auflage

Die Version 10.1 stellt wieder neue Funktionen zur Verfügung, welche in diesem Buch teilweise mit Beispielen demonstriert werden. Auch hier werden die neuen Beispiele durch das Präfix [10.1] markiert.

VORWORT

[10.2] Vierte Auflage

Zum Release der Java Client Version 10.2 wurde das Dokument noch mal erweitert und überarbeitet. Die neuen Beispiele werden wieder durch das Präfix [10.2] markiert.

[11.0] FÜNFTE AUFLAGE

Mit dem Release der Version 11.0 wurde auch dieses Dokument noch mal erweitert. Da sich die Versionen 10.2 und 11.0 nicht so sehr unterscheiden, gibt es nur eine Handvoll neuer Beispiele.

[12.0] SECHSTE AUFLAGE

Ein paar weitere Beispiele zu der neuen Version.

Matthias Thiele
Geschäftsführung ELO Digital Office GmbH
30 Mai 2018

Workflow

1-CLICK WORKFLOW-START

An manchen Arbeitsplätzen gibt es bestimmte Workflows, die immer wieder gestartet werden müssen. In diesem Fall ist er unnötig aufwändig, wenn der Anwender zuerst im Tab Aufgaben den Button „Workflow starten“ drücken muss, dort dann aus der möglicherweise umfangreichen Liste aller Workflowtemplates das Richtige auswählt und anschließend den Namen quittiert.

Einfacher wäre es, wenn er diesen speziellen Workflow mit einem Klick starten könnte. Das ist über ein kleines Skript machbar.

```
const WORKFLOWTEMPLATE = 631;

function getScriptButton100Name() {
  return "Projektantrag";
}

function getScriptButtonPositions() {
  return "100,home,navigation";
}

function eloScriptButton100Start(){
  var view = workspace.activeView;
  if (!view.hasSelection()) {
    var title = "Workflowstart";
    var message = "<html><h3>Kein Eintrag ausgewählt</h3>Bitte wählen Sie zuerst einen
Eintrag aus zu dem der Workflow gestartet werden soll.</html>";
    workspace.showAlertBox(title, message);
    return;
  }

  var items = view.allSelected;
  while (items.hasMoreElements()) {
    var item = items.nextElement();
    var flowName = item.name;
    ixc.startWorkFlow(WORKFLOWTEMPLATE, flowName, item.sord.id);
  }
```

```
  var fbMessage = (view.selectionCount == 1) ?
                "Der Workflow wurde gestartet" :
                view.selectionCount + " Workflows wurden gestartet";
  workspace.setFeedbackMessage(fbMessage);
}
```

Das Skript legt sich einen Button „Projektantrag“ im Ribbon an. Wenn der Anwender diesen Klickt, wird der über die Konstante WORKFLOWTEMPLATE voreingestellte Workflow gestartet. Als Name wird der Name des Dokuments verwendet.

Man sieht an dem Beispiel auch, dass Skripte in vielen Fällen leicht eine Mehrfachselektion unterstützen können. Statt einfach nur einen Eintrag aus firstSelected zu holen, wird die komplette Liste mittels allSelected gelesen und in einer Schleife abgearbeitet. Wenn man das User Interface schön machen möchte, kann man den Meldungstext noch an die Anzahl anpassen.

AUTOMATISCHE BENACHRICHTIGUNG BEI WORKFLOWAUFGABEN

Wenn ein Anwender nicht regelmäßig mit ELO arbeitet, wird er nicht schnell genug erkennen, wenn er eine neue Workflowaufgabe erhalten hat. Deshalb gibt es eine ELOas Libray, die automatisiert Benachrichtigungen versenden kann. Zur Verwendung muss man nur eine einfache Rule erstellen, die die Prüfung durchführt und die Mails versendet. Über das Intervall der Rule kann man festlegen, wie schnell der Anwender benachrichtigt werden soll.

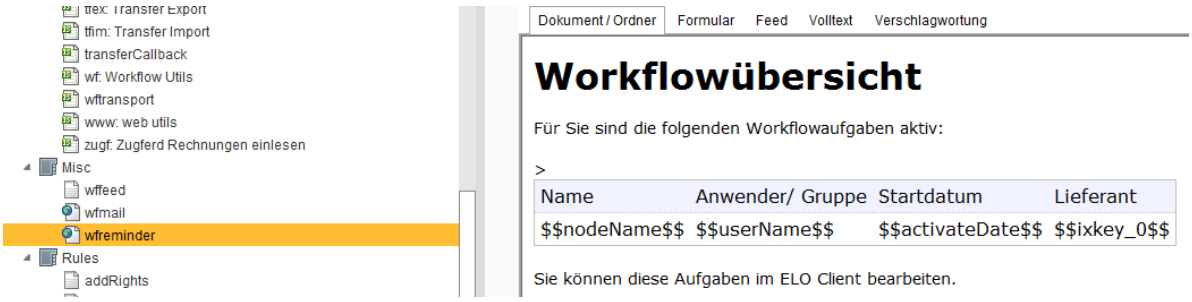
In der Rule muss man im Wesentlichen nur einstellen, über welchen Mailserver die Benachrichtigung verschickt werden soll und wie die Absender-Adresse lautet. Die Empfängeradresse wird aus dem Mail-Addressfeld des ELO Anwenders ausgelesen.

```
<ruleset>
  <base>
    <name>NotifyWf</name>
    <search>
      <name>"DIRECT"</name>
      <value>"1"</value>
      <mask>0</mask>
      <max>200</max>
    </search>
    <interval>6:00</interval>
  </base>
<rule>
```

```
  <name>Regel1</name>
  <condition></condition>
  <script>mail.setSmtpHost("MyMailServer");

notify.processAllUsers("Support@myCompany.com", "Benachrichtigung", true, true, true);
</script>
</rule>
<rule>
  <name>Global Error Rule</name>
  <condition>OnError</condition>
  <script></script>
</rule>
</ruleset>
```

Im ELOas Base Unterordner Misc gibt es für den Mailversand ein HTML Template, welches den Text der Mail enthält.




```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Workflowübersicht</title>
    <style type="text/css" >
      body {
        font-family: Verdana,Arial;
        font-size: 14px;
      }

      table {
        margin-top: 20px;
        margin-bottom: 20px;
```

```
      border: 1px silver solid;
      border-collapse: collapse;
    }

    td {
      border-bottom:1px silver dotted;
      padding: 5px;
    }

    .header {
      background-color:#f0f2ff;
    }

    .urgent {
      background-color:#ffd0d0;
    }

    .group {
      background-color:#d0ffd0;
    }

  </style>
 </head>
 <body>
  <h1>Workflowübersicht</h1>
  Für Sie sind die folgenden Workflowaufgaben aktiv:
  <table border="0">
  <tr><td class="header">Name</td><td class="header">Anwender/ Gruppe</td>
                        <td class="header">Startdatum</td><td
class="header">Lieferant</td></tr>
  <!--ListStart-->
  <tr><td class="$$className$$">$$nodeName$$</td><td
class="$$className$$">$$userName$$</td><td
class="$$className$$">$$activateDate$$</td>
  <td class="$$className$$">$$ixkey_0$$</td></tr>
```

```
    <!--ListEnd-->
    </table>
    Sie können diese Aufgaben im ELO Client bearbeiten.
  </body>
</html>
```

Jeder Anwender kann persönlich entscheiden, ob der die Benachrichtigung zugeschickt bekommen soll oder nicht. Hierzu müssen die jeweiligen Einstellungen in den profileopts des Anwenders gespeichert werden. Diese kann man per Java Client Skript ToggleMailReport auslesen und schreiben. Ein Dialog stellt den aktuellen Zustand dar und bietet dem Anwender die notwendigen Einstellmöglichkeiten.

```
function getScriptButton513Name() {
  return getText("ScriptButtonName");
}

function getScriptButtonPositions() {
  return "513,home,view";
}

function eloScriptButton513Start(){
  var actOpt = archive.getUserOption("ELOas.SendWfAsMail", "");
  actOpt = selectOptions(actOpt);
  if (actOpt >= 0) {
    archive.setUserOption("ELOas.SendWfAsMail", actOpt);
  }
}

function selectOptions(actOptions) {
  var dlg = workspace.createGridDialog(getText("Title"), 1, 8);
  var panel = dlg.gridPanel;

  actOptions = Number(actOptions);

  var ckMail = panel.addCheckBox(1, 1, 1, getText("Activate"), (actOptions & 1) != 0);
```

```
  var labelText = '<html><div style="width:400px; padding:5px; background-color: #f0f0f0">' +
        getText("OptionSeparator") +
        '</div></html>';

  var label = panel.addLabel(1,3,1, labelText);

  var ckAlways = panel.addCheckBox(1, 4, 1, getText("SendAlways"), (actOptions & 2) != 0);
  var ckGroup = panel.addCheckBox(1, 5, 1, getText("SendGroup"), (actOptions & 4) != 0);
  var ckDeputy = panel.addCheckBox(1, 6, 1, getText("SendSubstitute"), (actOptions & 8) !=
0);
  var ckWeekend = panel.addCheckBox(1, 7, 1, getText("WithWeekend"), (actOptions & 16) !=
0);
  var ckOnlyOnce = panel.addCheckBox(1, 8, 1, getText("SendOnce"), (actOptions & 32) !=
0);

  var result = -1;
  if (dlg.show()) {
    result = 0;

    if (ckMail.isChecked()) {
      result |= 1;
    }

    if (ckAlways.isChecked()) {
      result |= 2;
    }

    if (ckGroup.isChecked()) {
      result |= 4;
    }

    if (ckDeputy.isChecked()) {
      result |= 8;
    }
```

```
    if (ckWeekend.isChecked()) {
      result |= 16;
    }

    if (ckOnlyOnce.isChecked()) {
      result |= 32;
    }
  }

  return result;
}

function getText(name) {
  return utils.getText("ToggleMailReport", name);
}
```

Dieses Skript benötigt noch eine externe Textdatei „text_ToggleMailReport_DE“ für die Übersetzung des User Interface.

```
ScriptButtonName=Mail Benachrichtigung
Title=Einstellungen - Mail Benachrichtigung
Activate=EMail Benachrichtigung aktivieren.
OptionSeparator=Einstellungen zur Workflow-Benachrichtigung
SendAlways=Immer benachrichtigen, auch wenn keine Termine aktiv sind.
SendGroup=Auch bei Gruppenterminen benachrichtigen.
SendSubstitute=Auch bei Vertretungsterminen benachrichtigen.
WithWeekend=Auch am Wochenende benachrichtigen.
SendOnce=Pro Workflow nur einmal benachrichtigen.
```

MAIL AUS EINEM WORKFLOW HERAUS VERSENDEN

Das ELOas Modul notify besitzt die Möglichkeit einen Workflowknoten darauf zu überprüfen, ob eine Mail verschickt werden soll. Damit diese Prüfung regelmäßig durchgeführt wird, muss eine einfach Rule in einem kurzen Intervall (1 bis 60 Minuten) eingerichtet werden, die dann die Methode notify.checkSendMail aufruft.

```
<ruleset>
<base>
<name>SendWfMail</name>
```

```
<search>
<name>"WORKFLOW"</name>
<value>"1"</value>
<mask>0</mask>
<max>200</max>
</search>
<interval>1M</interval>
<onstart>
  EM_ALLOWALLMASKS = true;
</onstart>
</base>
<rule>
<name>Send</name>
<condition></condition>
<script>
  mail.setSmtpHost("MyMailServer");
  notify.checkAddFeed();
  notify.checkSendMail();
</script>
</rule>
<rule>
<name>Global Error Rule</name>
<condition>OnError</condition>
<script></script>
</rule>
</ruleset>
```

Wenn man aus einem Workflow heraus einen Mailversand auslösen möchte, muss man dort einen ELOas Knoten einfügen, der die notwendigen Informationen zu der Mail enthält.

Diese Daten werden im Bemerkungsfeld des Knotens eingetragen. Es muss in der ersten Zeile den Text #wfsendmail enthalten. Danach folgen weitere Parameter zum Empfänger (der z.B. aus einer Indexzeile ausgelesen werden kann), zum verwendeten Mail-Template (welches im ELOas Base\Misc Ordner liegen muss) oder zum Betreff. Man kann auch angeben, dass das aktuelle Dokument als Attachment mitgesendet wird.

Die vollständige Parameterliste finden Sie in der Beschreibung des notify Projekts.

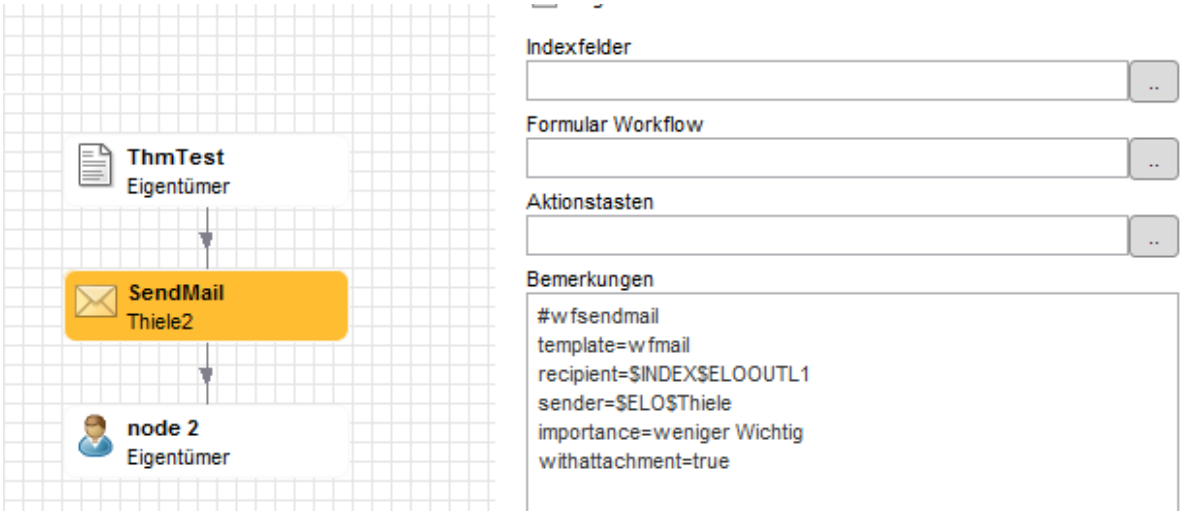


Abbildung 1: Knotendefinition zum Mailversand

Über den gleichen Mechanismus kann ein Workflow auch einen Feed Eintrag erzeugen. Es ist dafür kein Skript notwendig. Das Bemerkungsfeld muss in diesem Fall mit #wfaddfeed oder #wfmailandfeed beginnen. Über den Parameter feedtemplate kann man den Namen der Vorlage festlegen, wenn der Parameter nicht vorhanden ist, wird das default-Template wffeed verwendet.

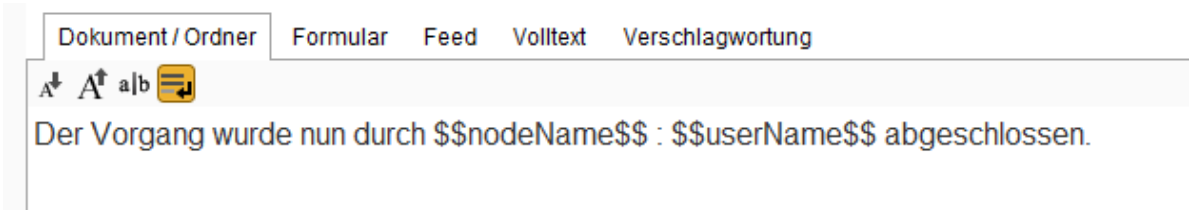


Abbildung 2: Template wffeed

Die Variablen aus dem Template werden dann durch die aktuellen Werte ersetzt und der erzeugte Text in den Feed des aktuellen Dokuments geschrieben.

Eine vollständige Beschreibung der Templates und Variablen finden Sie in der notify Dokumentation.

WORKFLOW IM JAVA CLIENT WEITERLEITEN

Wenn man eine Skriptaktion auf einer Workflowaufgabe durchführt, die verschiedene Tätigkeiten ausführt und anschließend den Workflow ohne Anwendereingriff automatisch weiterleiten soll, kann man es so erreichen:

```
var task = workspace.activeView.firstSelected;
if (task.isWorkflow()) {
  var nextNodeId = 1;
  task.confirmFlow([nextNodeId]);
```

```
}
```

Der Parameter in confirmFlow enthält eine Liste von Knoten-Ids (nicht Workflow-Id, die werden hier nicht benötigt, da man ohnehin nur innerhalb des Workflows weiterleiten kann).

Beachten Sie bitte, dass die Methode confirmFlow nur für Workflow-Einträge angeboten wird, nicht für Wiedervorlagetermine.

WORKFLOWAUFGABE PER MAIL ANKÜNDIGEN

Dieses Indexserverskript sendet eine Mail wenn ein Anwender einen neuen Termin von einem anderen Anwender erhält. Im Normalfall ist die Notify Funktion des ELOas besser geeignet, da sie alle vorhandenen Termine eines Anwenders zu einer Mail zusammenfasst. In speziellen Fällen ist es aber notwendig, dass die Benachrichtigung sofort und für jeden einzelnen Termin gesendet wird.

Dafür muss im Tomcat des Indexservers der Mailserver konfiguriert werden (siehe IX Buch von Herrn Imig) und im Workflow im Personenknoten Start-Event dieses Indexserver-Skript eingetragen werden (siehe Attachment). Die Mail wird nur erzeugt, wenn die Aufgabe von einer anderen Person kommt. Aufgaben, die man sich selber erzeugt, lösen keine Mail aus. Und natürlich werden die Mails nur von Indexserver Clients erzeugt, nicht vom Windows Client.

```
function onEnterNode( ci, userId, workflow, nodeId ){
  try {
    var node = getNodeById(workflow, nodeId);
    if (node && (userId != node.userId)) {
      var mail = getUserAddress(ci, node.userId);
      var subject = "Workflowaufgabe: " + node.name;
      var body = '<h1>' + node.name + '</h1><h4>' + workflow.name + '</h4><a href =
"elodms://wf/' + workflow.id + '/' + node.id + '">Sie haben eine neue Workflowaufgabe
erhalten.</a>';
      sendMail(mail, subject, body);
    }
  } catch(ex) {
    log.info("Error processing Workflow Info: " + ex);
  }
}

function onAfterCheckinReminder(ec, reminders, sord, sordz, lockz) {
  for (var i = 0; i < reminders.length; i++) {
    var reminder = reminders[i];
    log.debug( "Reminder: " + reminder.name );
```

```
    var receiverId = reminder.receiverId;
    if (ec.user.id != receiverId) {
      try {
        var mail = getUserAddress(ec.ci, receiverId);
        log.debug("UserName: " + reminder.receiverName + ", Mail: " + mail);
        if (mail) {
          var subject = "Wiedervorlage: " + reminder.name;
          var body = "<h1>" + reminder.name + '</h1><a href="elodms://' + sord.guid + '">Sie
haben einen neuen Wiedervorlagetermin erhalten.</a>';
          sendMail(mail, subject, body);
        }
      } catch(ex) {
        log.info("Error processing Reminder Info: " + ex);
      }
    }
  }
}

function sendMail(to, subject, body) {
  var sendMail = new Packages.de.elo.ix.jscript.SendMail("mail/SRVPEMAIL02vm");
  sendMail.subject = subject;
  sendMail.sender = "eloservice@elo.com";
  sendMail.TO = to;
  sendMail.body = body;
  sendMail.send();
}

function getUserAddress(ci, userId) {
  var userData = ix.checkoutUsers(ci, [userId], CheckoutUsersC.BY_IDS_RAW, LockC.NO);
  var mail = userData[0].userProps[UserInfoC.PROP_NAME_EMAIL];
  log.debug("Mail address of user " + userId + " is " + mail);

  return mail;
}
```

```
function getNodeById(workflow, nodeId) {
  var nodes = workflow.nodes;
  for (var i = 0; i < nodes.length; i++) {
    var node = nodes[i];
    if (node.id == nodeId) {
      return node;
    }
  }

  return null;
}
```

WORKFLOWSTATUS PER SKRIPT ÄNDERN

Der Workflowstatus wird im Startknoten im Condition Feld gespeichert. Ab der Version 9.3 besitzt das WorkflowElement eine Methode, die das in einem Schritt erledigt.

```
workspace.activeView.firstSelected.workflowStatus = "aus dem Skript";
```

WORKFLOWKNOTEN ENTZIEHEN

Wenn eine Workflowaufgabe parallel zur Freigabe an mehrere Anwender geht, ist es sinnvoll, die Verarbeitung für alle betroffenen Anwender abzubrechen, sobald auch nur ein Anwender abgelehnt hat.

Zur Realisierung wird für den Zweig „Freigabe nicht erteilt“ ein Sammelknoten angelegt, der bereits beim ersten abgeschlossenen Vorgängerknoten weitergeleitet wird. Im Ende-Skript werden dann alle Freigabe-Anwenderknoten zurückgesetzt.

Dieses Skript wird hier nicht aufgeführt, da es nicht mehr notwendig ist. Die Liste der Knoten zum Zurücksetzen wird im Sammelknoten eingetragen. Die Skriptfunktion ist fest in den Indexserver eingebaut.

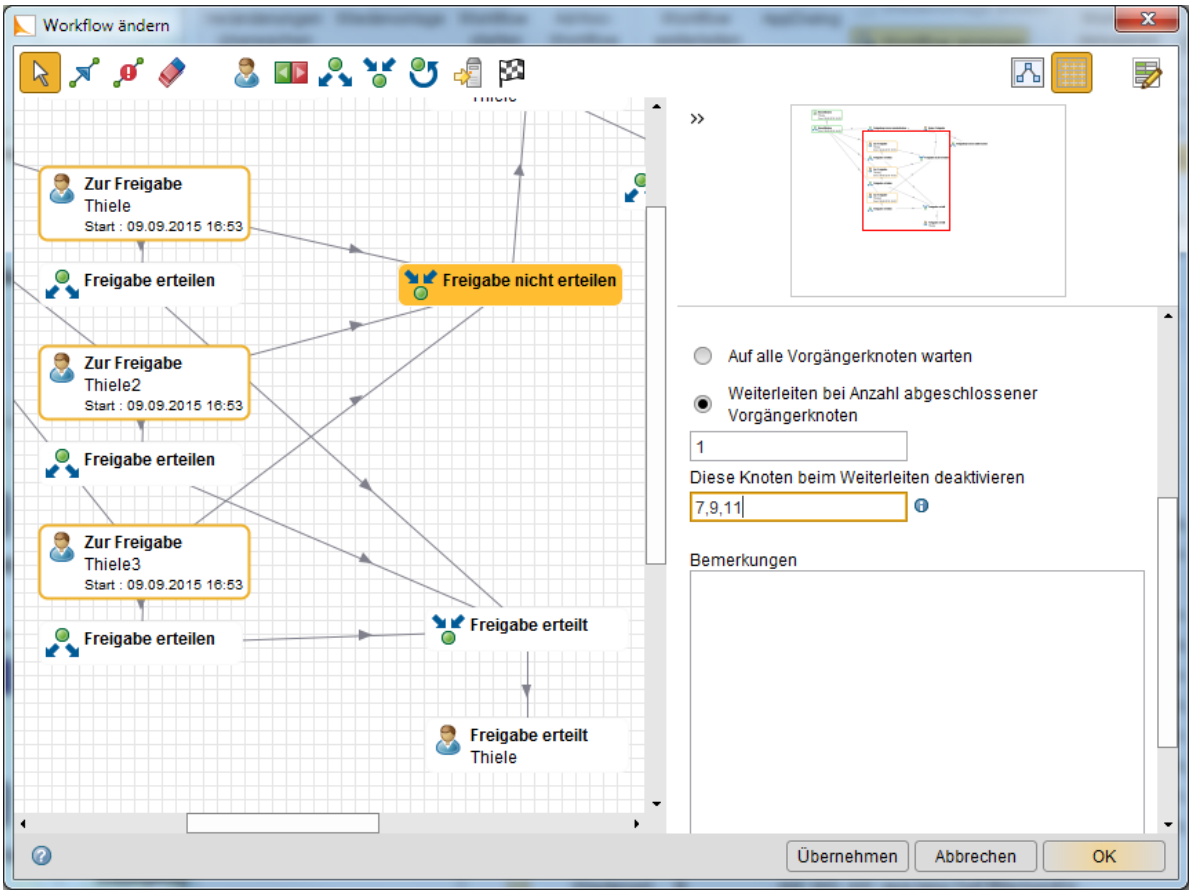


Abbildung 3: Sammelknoten mit Rücksetzliste

WORKFLOW-WEITERLEITEN ACTION BUTTONS

In der Workflowdefinition kann man zu jedem Personenknoten bis zu 5 Aktions-Buttons hinterlegen. Diese werden im dem Weiterleiten-Dialog angezeigt und rufen eine vordefinierte Skriptfunktion auf.

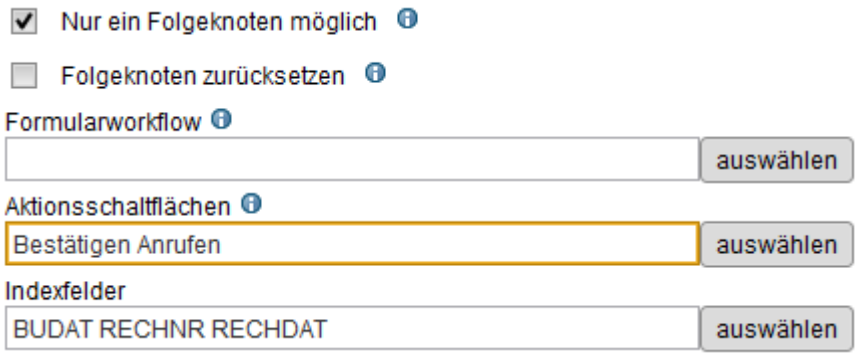


Abbildung 4: Aktionsbuttons definieren

Zu jedem Aktionsbutton muss eine Skriptfunktion hinterlegt werden, wessen Name aus einem Präfix und der Button-Bezeichnung besteht: cfb<Name>Start()

```
function cfbBesttigenStart() {
  workspace.showAlertBox("Workflow Action", "Der Anwender hat den Button 'Bestätigen'
gedrückt");
}
```

Wenn dieser Button beim Weiterleiten geklickt wird, ruft der Java Client die dazu passende Skriptfunktion auf.

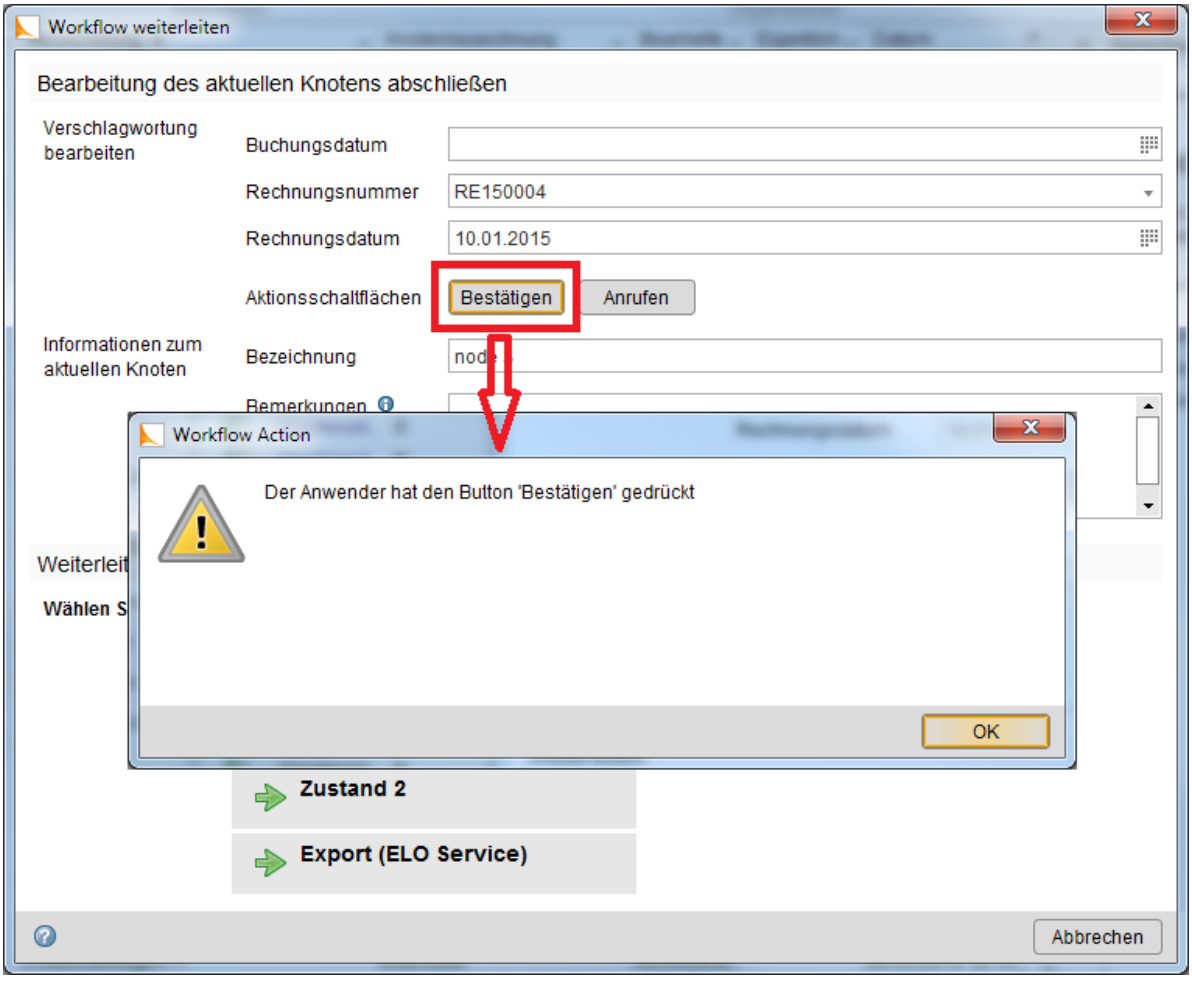


Abbildung 5: Aktive Skriptfunktion

Da der Aktionsname direkt in einen JavaScript Funktionsname einfließt, dürfen diese Namen nur aus den Buchstaben A bis Z, Ziffern und dem Unterstrich bestehen. Alle anderen Zeichen erzeugen in der Version 9 einen Syntaxfehler.

Ab der Version 10.0 sind auch andere Zeichen zulässig. Diese werden aus dem Funktionsnamen ausgefiltert. Man muss bei der Benennung der Funktion die gleichen Zeichen entfernen, sonst findet der Aufruf nicht die dazu gehörende Event-Funktion: „Bestätigen -> Besttigen“

[10] SKRIPT-EVENT BEIM AKTUALISIEREN DER AUFGABENLISTE

Ab der Version 10 kann man über ein Skript-Event auf neue Aufgaben in der Aufgabenliste reagieren. Beachten Sie bitte, dass beim Programmstart zwar alle Aufgaben „neue Aufgaben“ sind - das Skript-Event aber genau aus diesem Grund hier noch nicht aktiviert wird. Erst bei der Aktualisierung der bestehenden Daten wird das Event aktiv.

Als Parameter werden dem Event das Indexserver UserTask Objekt (Workflow/ Wiedervorlage/ Aktivität) sowie die Information, ob der Anwender die Aufgabe bereits vorher mal gesehen hat.

```
function eloNewTaskAvailable(task, isSeen) {
  workspace.showAlertBox(isSeen, task.toString());
}
```

[10] WORKFLOW-DIAGRAMM PER SKRIPT BEARBEITEN

Zu einem WorkflowElement in der Tasks-Liste kann man leicht das komplette Workflow-Diagramm einlesen. Hierzu gab es schon immer den Befehl workflowItem.getDiagram(). Dieser liest das Workflow-Diagramm mit einer Sperre ein. Das hat gleich zwei wichtige Folgen:

1. Wenn das Diagramm bereits durch einen anderen Anwender oder Prozess gesperrt ist, schlägt der Aufruf mit einer Exception fehl. Das Skript muss damit rechnen und passend reagieren. Speziell beim Workflow Weiterleiten ist das Diagramm temporär kurz gesperrt. Es kann zu schwer identifizierbaren Problemen führen, wenn man das nicht weiß.
2. Nach Beendigung der Arbeiten muss die Sperre unbedingt wieder gelöst werden. Sonst bleibt der Workflow dauerhaft gesperrt und kann nicht mehr weitergeleitet werden.

Wird der Workflow nur lesend benötigt, kann man ihn über den Befehl workflowItem.getDiagramWithoutLock() ohne Sperre einlesen. Dieser Befehl scheitert nicht an einer anderen Sperre und erzeugt auch keine Sperre für andere Prozesse. Sie sollten diesen Workflow dann aber auf keinen Fall bearbeiten und zurückspeichern. Falls in der Zwischenzeit andere Änderungen vorgenommen wurden, würden Sie diese auf nicht vorhersagbare Art und Weise teilweise Rückgängig machen und somit einen inkonsistenten Zustand auslösen.

[10] PARENT-WORKFLOW EINES SUB-WORKFLOWS ERMITTELN

Wenn man aus einem Sub-Workflow auf den Parent-Workflow zugreifen möchte, aus dem der Sub-Workflow gestartet wurde, finden Sie sowohl im WFDiagram wie auch im WFCollectNode das Property parentFlowId.

```
function eloHomeStart() {
  var item = workspace.activeView.firstSelected;

  if ( item.isWorkflow() ) {
    var flowNode = item.task.wfNode;
    var parentFlow = flowNode.parentFlowId;

    workspace.showInfoBox( "SubFlow", flowNode.nodeName + " : " + parentFlow );

    if (parentFlow > 0) {
      var parentWorkflow = ixc.checkoutWorkFlow( parentFlow, WFTypeC.ACTIVE,
                                                 WFDiagramC.mbAll, LockC.NO );

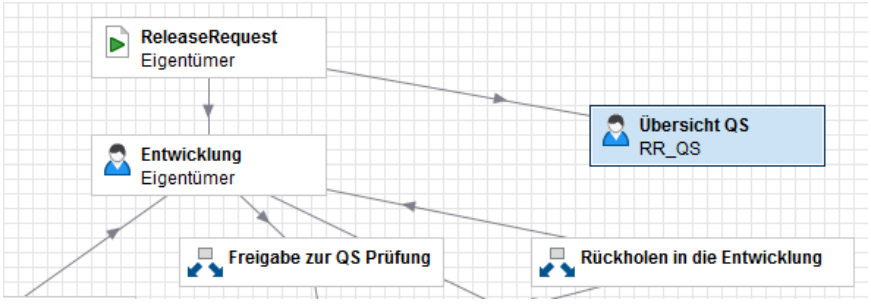
      workspace.showInfoBox( "ParentFlow", parentWorkflow.name );
    }
  }

  return -1;
}
```

Der Datentransfer zwischen den beiden Workflows ist einfach. Da beide Workflows auf dem gleichen Objekt laufen, teilen sie sich das gleiche Sord Objekt und Sord-Map Objekt mit der Verschlagwortung. Lediglich der Zugriff auf die WF-Map Daten ist etwas aufwändiger, hier muss ein checkoutMap auf die Parent-Workflow-Id statt der eigenen Workflow-Id erfolgen.

[10.1] Workflow-Annahme verhindern (Indexserver)

Wenn man einer Gruppe einen Überblick über alle laufenden Instanzen eines Workflow-Templates geben möchte, kann man einen zusätzlichen Knoten für diese Gruppe an den Startknoten anhängen. Dieser stellt sicher, dass die Gruppe über die komplette Laufzeit des Workflows einen aktiven Knoten besitzt. Somit kann jeder Mitarbeiter dieser Gruppe alle Workflows einsehen.



Das Problem bei dieser Vorgehensweise liegt darin, dass irgendwann immer ein Mitarbeiter diesen Knoten irrtümlich annimmt. Damit ist er nicht mehr der Gruppe zugeordnet und die anderen Mitglieder verlieren diesen zusätzlichen Zugriff.

Mittels eines kleinen Indexserverskripts kann man sicherstellen, dass nur eine bestimmte Person diesen Knoten annehmen kann, z.B. der Teamleiter. Wenn der Workflow einen ausdrücklichen Ende Knoten besitzt, ist es nicht notwendig, den Knoten anzunehmen und weiterzuleiten. Wenn es aber keinen Ende Knoten gibt, muss der Teamleiter nach Abschluss aller Arbeiten, diesen Knoten annehmen und weiterleiten, damit der Workflow geschlossen werden kann.

```
function onBeforeTakeWorkFlowNode(ec, workflow, node, user, flags, lock) {
  if ((workflow.templateName == "ReleaseRequest") && (node.id == 30)) {
    if (user.name != "HeadOfQsDepartment") {
      log.warn("User tried to take QS node: " + user.name);
      throw "Dieser Knoten kann nur von der QS Leitung angenommen werden."
    }
  }
}
```

Java Client allgemein

SKRIPTE IN MEHREREN SPRACHEN

Ein Skript besitzt oft Texte zur Anzeige, die in unterschiedliche Sprachen übersetzt werden müssen, wenn das Skript in mehreren Ländern eingesetzt werden soll. Theoretisch könnte man das Skript mehrfach kopieren und einfach die Texte jeweils austauschen. Das wäre aber sowohl von der Übersetzung her fehlerträchtig – woher soll der Übersetzen sicher wissen, welcher Text übersetzt werden muss und welcher nicht – als auch von der Wartung her schwierig. Bei einem Fehler müssten alle Versionen korrigiert werden und keine darf vergessen werden.

Deshalb besitzt der Java Client einen Mechanismus, der es erlaubt, die Anzeigetexte eines Skripts von dem eigentlichen Programmcode zu trennen. Die Texte werden hierzu in eine Textdatei ausgelagert. Der Name der Datei ist üblicherweise gleich zu dem Skriptdateinamen mit dem Präfix text_ und als Endung wird noch ein Länderkennzeichen angefügt (_DE oder _EN …).

Das Skript ToggleMailReport besitzt eine Textdatei mit dem Namen text_ToggleMailReport_DE für die deutschen Texte.

```
ScriptButtonName=Mail Benachrichtigung
Title=Einstellungen - Mail Benachrichtigung
```

Im Skript selber wird über eine utils-Funktion auf den Text zugegriffen.

```
var title = getText("Title");
var buttonName = getText("ScriptButtonName");


…
function getText(name) {
  return utils.getText("ToggleMailReport", name);
}
```

EXTERNE JAR DATEIEN IN SKRIPTE EINBINDEN

Der Java Client kann externe jar Dateien in Skripte einbinden. Das ist nützlich, wenn eine gewünschte Funktion bereits als Java Source Code oder fertiges jar Modul zur Verfügung steht und diese aus einem Skript heraus aufgerufen werden soll.

Zur Verwendung muss man nur die jar Datei (RoboMinerDemo.jar) mit in das Java Client Skript Verzeichnis packen. Es wird dann automatisch mit dem Client deployed und steht in den Skripten anschließend zur Verfügung. Als Beispiel wird hier ein kleines Demoprogramm aus dem Internet verwendet.

```
function getScriptButton100Name() {
  return "Robo Miner";
}

function getScriptButtonPositions() {
  return "100,home,view";
}

function eloScriptButton100Start(){
  com.rnet.robominer.RoboMinerDemo.main( [] );
}
```

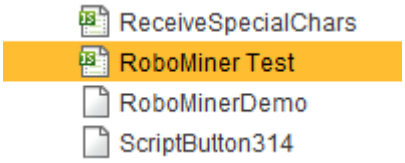


Abbildung 6: RoboMiner Test Skript und Demo jar Archiv im Skript Ordner

Dieses Beispiel zeigt auch gut die Risiken auf: Wenn man das RoboMiner Fenster schließt, führt das Programm ein System.exit() aus und beendet damit nicht nur sich selber, sondern gleich den kompletten Java Client. Alle Fehler in dem eingebundenen Java Programm schlagen hart auf den Client zurück.

EXTERNE PROGRAMME AUFRUFEN

Sie können aus einem Java Client Skript leicht ein externes Programm oder eine Browseranzeige aufrufen. Das passiert über die Java Runtime bzw. Desktop Klassen.

Programm aufrufen

```
function execute(command) {
    log.debug("Execute command: " + command);
    var p = Runtime.getRuntime().exec(command);
    p.waitFor();
    log.debug("Done.");
}
```

Alternativ für einen Aufruf ähnlich dem ShellExecute

```
Desktop.desktop.edit(file);
```

Browser aufrufen

```
function searchCustomer() {
  var name = indexDialog.getObjKeyValue(0);
  var codec = new URLCodec();
  var encodedName = codec.encode(name);
  var uri = new URI("http://www.google.de/search?q=" + encodedName)
  Desktop.desktop.browse(uri);
}
```

Manchmal kommt es vor, dass man auf das Ende des aufgerufenen Programms warten muss. Hierfür gibt es Methode waitFor():

```
function eloScriptButton100Start(){
  var command = [ "C:\\Program Files (x86)\\Notepad++\\Notepad++.exe", "d:\\temp\\test.txt" ];
  var runtime = new Packages.java.lang.Runtime.getRuntime();
  var process = runtime.exec(command);
  process.waitFor();
  workspace.showInfoBox("Test", "Notepad++ wurde beendet");
}
```

HAFTNOTIZEN IN DER POSTBOX EINFÜGEN

Wenn man Haftnotizen (Randnotizen oder Annotations) per Skript in der Postbox einfügen will, muss man auf ein paar Stolperfallen achten, die daraus entstehen, dass der Viewer zur aktuellen Ansicht einen Cache mitführt, der auch regelmäßig geschrieben wird. Das kann leicht dazu führen, dass die eigene Änderung wieder rückgängig gemacht wird.

```
var item = workspace.activeView.firstSelected;
if (item != null) {
  // Ansicht löschen damit die neuen Notizen nicht wieder überschrieben werden
  intray.clearSelection();

  // Marker erzeugen
  var note = ixc.createNote2(0, NoteC.TYPE_ANNOTATION_MARKER, "");
  note.XPos = 100;
  note.YPos = 50;
  note.width = 200;
  note.height = 50;
```

```
  note.color = 0xc1ffc2;

  item.addNote(note);

  intray.selectDocument(item);
}
```

Der „Trick“ besteht darin, dass vor dem Einfügen die aktuelle Anzeige mittels clearSelection() gelöscht wird, damit sind alle Änderungen des Anwenders sicher im Indexserver bekannt und es gibt keinen lokalen Cache mehr, der alte Daten zum Überschreiben verwenden könnte.

Wenn man am Ende selectDocument() aufruft, wird die Anzeige wiederhergestellt und die neue Haftnotiz direkt angezeigt.

COMMANDLINK DIALOG

Wenn sich ein Anwender unter einer Reihe von möglichen Optionen entscheiden soll, stellt ELO einen zu Windows vergleichbar gestalteten Command Link Dialog zur Verfügung. Diese Auswahl ist ansprechender und schneller als ein Ok/Abbrechen Dialog mit einer Auswahlliste. Sinnvoll einsetzen kann man den Dialog allerdings nur bei einer überschaubaren Anzahl von Optionen.

Die Verwendung ist recht einfach. Man gibt eine Liste mit den Optionen und eine weitere Liste mit Beschreibungen zu den Optionen an und ruft den Dialog auf.

```
function eloScriptButton100Start(){
  var result = simpleSelection();
  workspace.setFeedbackMessage("Gewählte Option: " + result);
}

function simpleSelection() {
  var choices =
    ["Vollständige Liste",
     "Nur die erste Spalte",
     "Noch eine dritte Option"];

  var descriptions =
    ["Das passiert, wenn Sie 'vollständige Liste' auswählen",
     "Und das passiert bei 'Nur die erste Spalte'",
     "Das passiert, wenn Sie die dritte Option auswählen"];
```

```
  var result = workspace.showCommandLinkDialog("Bereichsliste erstellen",
    "Wählen Sie aus, welche Bereichsliste Sie erstellen möchten",
    "", CONSTANTS.DIALOG_ICON.QUESTION,
    choices, descriptions, []);

  return result;
}
```

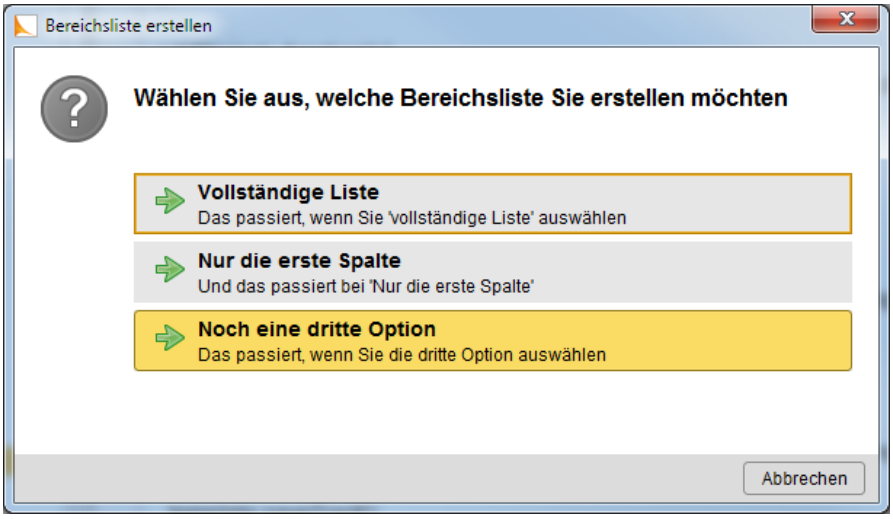


Abbildung 7: CommandLink Dialog mit drei Optionen

COMMANDLINK DIALOG AUS EINER ORDNERAUSWAHL

Es kommt nicht selten vor, dass der Anwender zwischen unterschiedlichen Arbeitsbereichen auswählen muss, die in Form einer ELO Ordnerliste vorliegen. Diese kleine Hilfsfunktion bietet einen sehr einfachen Weg, aus einer Ordnerliste einen CommandLink-Dialog zu erzeugen. Man muss nur den Parent Eintrag, den Titel und einen Bearbeitungshinweis angeben und die Hilfsfunktion zeigt die passende Auswahl an. Wenn der Anwender einen Archiveintrag auswählt, wird dieser als Ergebnis zurückgeliefert.

```
function eloScriptButton100Start(){
  var title = "Wählen Sie den gewünschten Arbeitsbereich aus";
  var message = "Die Bearbeitung kann in verschiedenen Bereichen durchgeführt werden.
Hier können Sie den passenden Bereich auswählen.";
  var parent = workspace.activeView.firstSelected;
  var result = childSelection(title, message, parent);
  workspace.setFeedbackMessage("Gewählte Option: " + result);
```

```
}
```

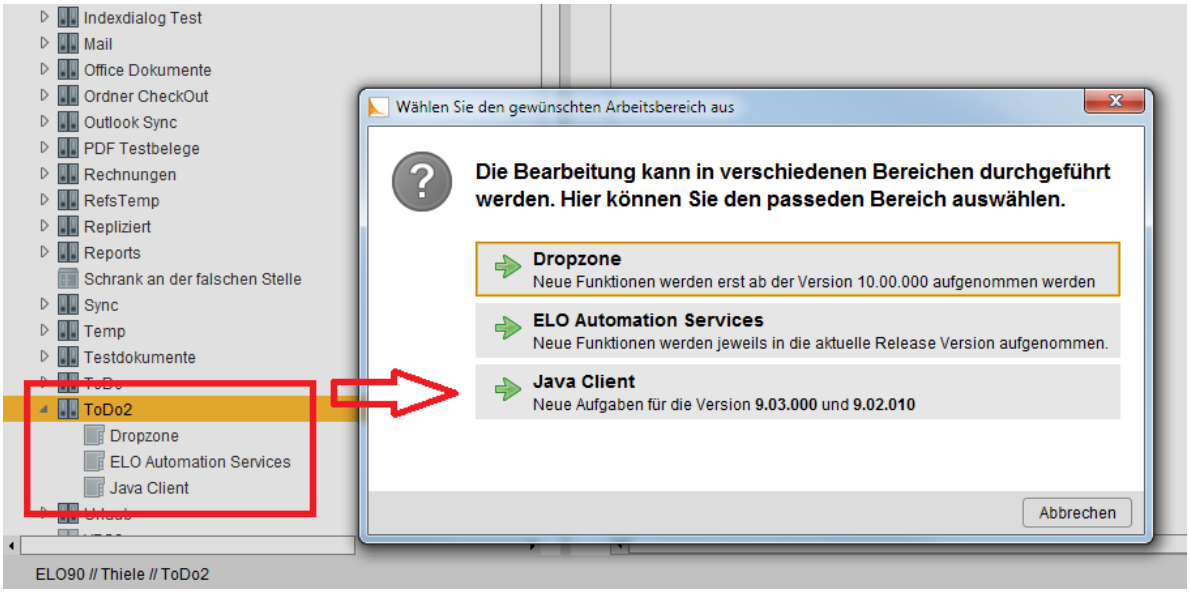


Abbildung 8: CommandLink Dialog mit Ordnerauswahl

Die benötigte Hilfsfunktion ist ebenfalls recht einfach gehalten:

```
function childSelection(title, message, parent) {
  var optionNames = new Array();
  var optionTexts = new Array();
  var optionElements = new Array();

  var items = parent.children;
  while (items.hasMoreElements()) {
    var item = items.nextElement();

    optionNames.push(item.name);

    var desc = item.sord.desc;
    if (desc && desc.length() && desc.startsWith("<html>")) {
      optionTexts.push(item.sord.desc);
    } else {
      optionTexts.push("");
    }
```

```
    optionElements.push(item);
  }

  var result = workspace.showCommandLinkDialog( title, message, "",
    CONSTANTS.DIALOG_ICON.QUESTION, optionNames, optionTexts, []);

  if (result > 0) {
    return optionElements[result - 1];
  } else {
    return null;
  }
}
```

ANNOTATIONS VERLINKEN

Der Java Client bietet die Möglichkeit, von einer Annotation auf eine andere verlinkte Annotation zu springen. Damit kann man sich Hyperlinks zwischen unterschiedlichen Dokumenten oder auch innerhalb eines Dokuments erstellen. So eine Verlinkung erstellt man, indem zuerst die Ziel-Annotation selektiert wird und über das Kontextmenü „Position merken“ aufgerufen wird.

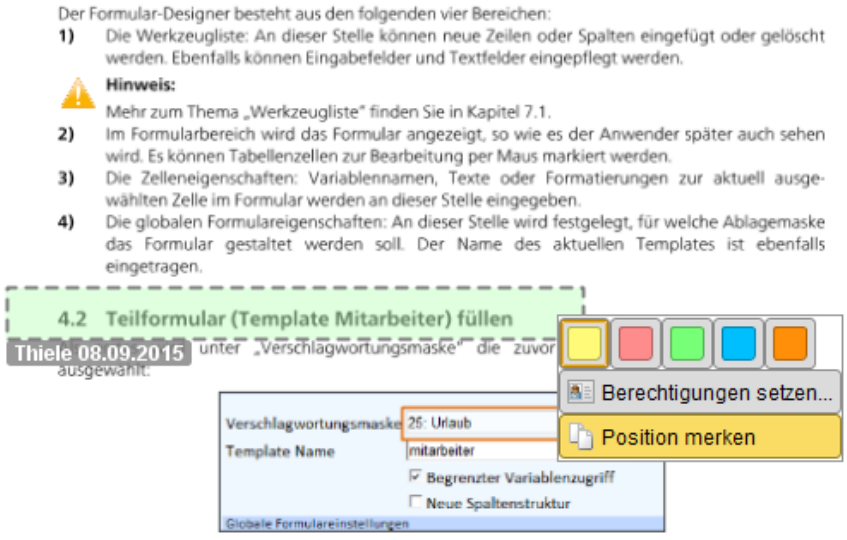


Abbildung 9: Position merken - das Ziel der Verlinkung

Anschließend kann man zum Startpunkt der Verlinkung gehen und dort die gemerkte Position in eine Textnotiz einfügen.

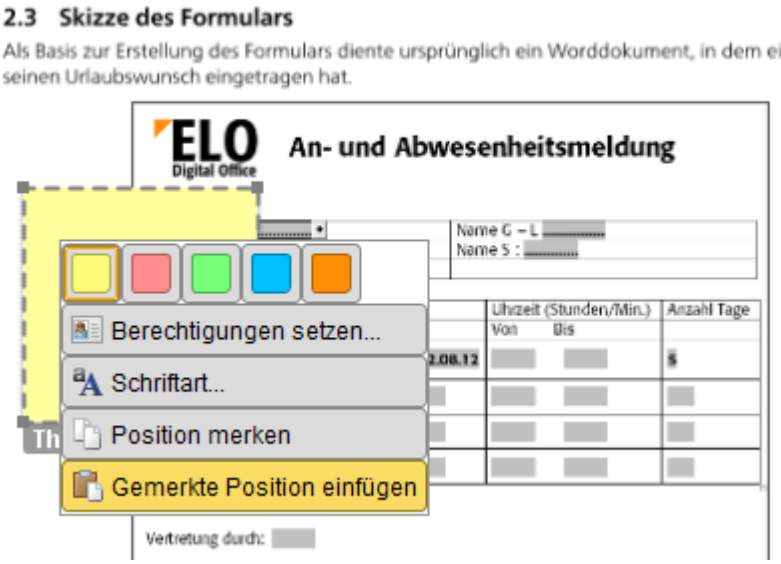


Abbildung 10: Gemerkte Position einfügen - Startpunkt festlegen

Im Standard wird in die Textnotiz der Name des Dokuments eingetragen. Man kann den Text aber auch per Skript verändern.

Von dieser Textnotiz kann der Anwender dann jederzeit über das Kontextmenü zu der verlinken Zielnotiz springen.

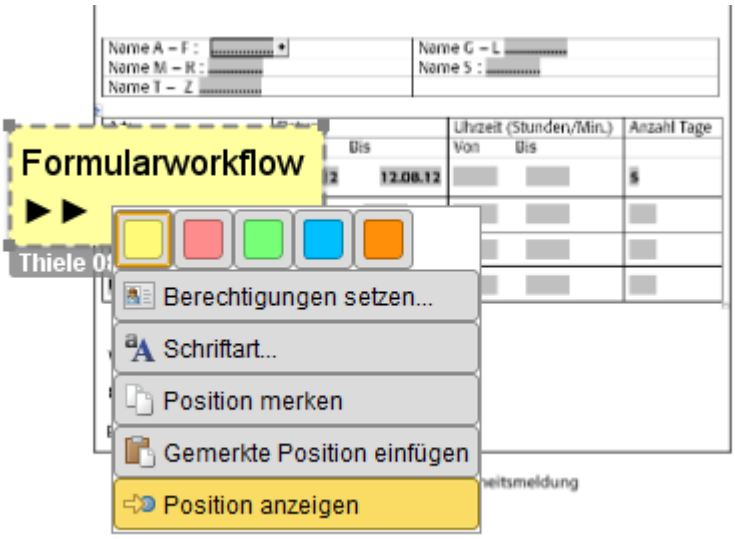


Abbildung 11: Gehe zum verlinkten Ziel

Zur Bearbeitung stehen zwei Skript-Events zur Verfügung. Einmal eloPrepareNoteLink – dieses Event wird aufgerufen, wenn der Anwender das Ziel markiert hat aber, noch bevor der eigentliche Link

erzeugt wird. An dieser Stelle kann man dann auch den Text beeinflussen, der später in die Textnotiz eingetragen wird.

```
function eloPrepareNoteLink( linkPosition, sord, note ) {
  var page = "Seite " + note.pageNo;
  linkPosition.elemName = page + "\n" + sord.name;
}
```

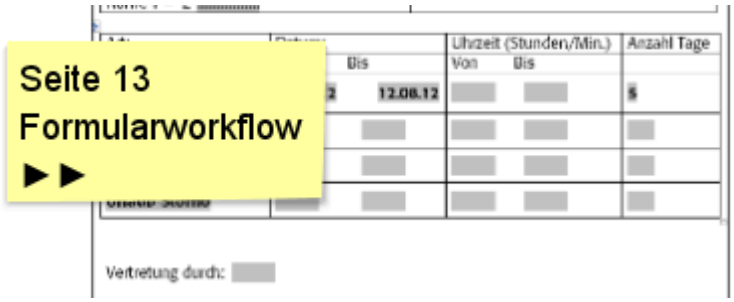


Abbildung 12: Erweiterter Text für die Verlinkung

Weiterhin steht das Event eloInsertNoteLink zur Verfügung. Dieses Event wird aufgerufen, wenn der Text in die Textnotiz eingetragen wird. Man kann den angezeigten Text also auch von der aktuellen Startposition abhängig machen.

```
function eloInsertNoteLink( linkPosition, sordId, note ) {
  var page = "Von Seite " + note.pageNo + " nach ";
  linkPosition.elemName = page + linkPosition.elemName;
}
```

2.3 Skizze des Formulars

Als Basis zur Erstellung des Formulars diente ursprünglich ein Worddokument, in dem ein Mitarbeiter seinen Urlaubswunsch eingetragen hat.

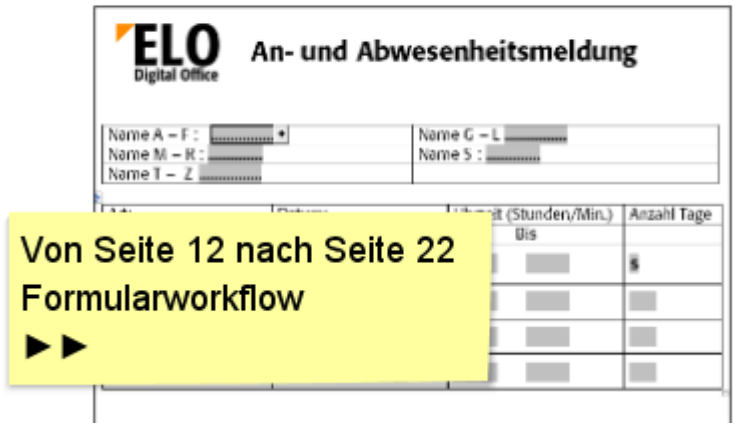


Abbildung 13: Anpassung am Start- und am Zielpunkt

VERWENDUNG DES HTTP AUTOMATION INTERFACE

Der Java Client besitzt ein Interface, welches über http Aufrufe angesprochen werden kann. Dafür ist im Java Client ein kleiner Webserver integriert, der diese Aufrufe entgegennimmt und an die Skriptschnittstelle weiterreicht.

Vorbereitung:

Die http Schnittstelle ist im Normalfall abgeschaltet. Sie muss über Profileopts-Einträge aktiviert werden. Dazu sind zwei Einträge notwendig - ein Boolean Wert, die sie einschaltet, und ein Integer Wert mit der Portnummer, auf der der Client auf Kommandos wartet.

Die beiden folgenden SQL Statements schalten die Schnittstelle für den User 0 (Administrator) ein. Sie können natürlich auch eine beliebige andere Anwender- oder Optionsgruppennummer sowie die globale Id verwenden.

```
insert into profileopts (userid, optkey, optvalue) values (0, 'EloJ.B.HAIServerPort', 1)
insert into profileopts (userid, optkey, optvalue) values (0, 'EloJ.S.HAIServerPort', 8990)
```

Der Java Client muss nach dieser Änderung neu gestartet werden da die Optionen nur beim Programmstart eingelesen werden.

Skript definieren

Damit über die http Schnittstelle nicht einfach beliebige Skripte aufgerufen werden können, ist sie so eingeschränkt, dass nur Skripte aktiviert werden, die mit den Namen http beginnen. Im Beispiel wird die Funktion httpDoSomething verwendet. Im Java Client wird ein Skript mit dieser Methode angelegt.

Die Methode erwartet zwei Parameter - die dann einfach als Feedback Message ausgegeben werden. Der Return Wert wird an den Aufrufer zurückgegeben:

```
function httpDoSomething(param1, param2) {
  workspace.setFeedbackMessage(param1 + " : " + param2);
  return "done";
}
```

Skript aufrufen

Der Aufrufer muss nun nur einen einfachen http GET Aufruf an localhost und die konfigurierte Portnummer senden. Das geht aus vielen Anwendungen heraus. Der Einfachheit halber verwende ich im Beispiel den Browser:

```
<a
href="http://localhost:8990/script/httpDoSomething/?param1=Something¶m2=More">
```

Der Aufruf löst im Client den Start des httpDoSomething Skripts aus und zeichnet "Something : More" auf den Bildschirm. Anschließend zeigt er die Rückgabe an.

```
OK|done
```

Das OK| signalisiert, dass der Aufruf erfolgreich war, done ist der Rückgabewert des Skripts.

Wenn man nur ein Kommando aufrufen will und die Seite nicht verlassen möchte, kann man das Kommando auch per AJAX Aufruf senden. Dafür definiert man sich im Browser Formular diese Funktion

```
function callScript(param1, param2) {
  var url = "http://localhost:8990/script/httpDoSomething/?param1=" + param1 + "¶m2=" +
param2;
  var xmlhttp=new XMLHttpRequest();
  xmlhttp.open("GET", url, true);
  xmlhttp.send();
}
```

Und der Aufruf sieht dann so aus:

```
<a  onclick="callScript('Something', 'More')">
```

Wenn man nicht nur ein Kommando absetzen möchte, sondern auch das Ergebnis braucht, muss man die xmlhttp.onreadystatechange Funktion entsprechend füllen.

Der große Vorteil des http Aufrufs gegenüber der COM Schnittstelle ist, dass dieser Aufruf keine lokale Installation benötigt. Er funktioniert deshalb auch beim WebStart Aufruf oder beim Start über die Batch Datei (in diesem Fall wird kein COM Objekt instanziiert). Zudem funktioniert dieser Aufruf auch unter Mac OS X und Linux und nicht nur unter Windows.

NACHRICHTEN ZWISCHEN UNTERSCHIEDLICHEN FENSTERN VERSENDEN

Eine eher exotische Anforderung – kommt aber hin und wieder mal vor: Wenn ein Anwender zwei ELO Ansichten offen hat, gibt es in jeder Ansicht einen eigenen Satz von Skripten. Ein Verwaltungsskript vom Fenster 1 weiß nichts vom Zustand des Fensters 2.

Damit man Informationen zwischen den Fenstern versenden kann, gibt es im Java Client eine Funktion, die so eine Möglichkeit bietet. Zum Versand muss einfach nur die Methode workspace.sendBroadcast aufgerufen werden. Hier legt man im ersten Parameter fest, an welches Fenster (oder an alle) die Nachricht geschickt werden soll. Im zweiten Parameter trägt man einen Nachrichtennamen und im dritten Parameter die eigentliche Nachricht ein.

Wenn ein Skript solche Broadcast Nachrichten empfangen möchte, muss es die Funktion eloBroadcast implementieren.

```
  function eloScriptButton103Start() {
    workspace.sendBroadcast(CONSTANTS.DESTINATION_WORKSPACE.ALL, "Neu", "My
Message");
  }

  function eloBroadcast(from, tag, message) {
    var msg = workspace.connectionNumber + " from " + from + " [" + tag + "] : " + message;
    workspace.setFeedbackMessage(msg);
  }
```

SKRIPT ÜBER DIE COM SCHNITTSTELLE AUFRUFEN

Im Windows Umfeld gibt es viele Programme, die andere Anwendungen über die COM Schnittstelle fernsteuern können. Der Java Client bietet auch eine einfache COM Schnittstelle mit einigen Funktionen an. In der Praxis ist es aber sinnvoll, sich auf eine einzige Funktion zu beschränken: RunScriptFunction.

Im Java Client hinterlegen Sie dazu ein Skript, welches die gewünschte Funktion bereitstellt. Der Funktionsname muss unbedingt mit der Zeichenfolge „eloCom“ beginnen. Andere Skripte können aus Sicherheitsgründen nicht über die externe COM Schnittstelle aufgerufen werden.

```
function eloComMyScript(param1, param2, param3) {
  var content = [
          "<h3>MyScript</h3>",
```

```
                "Param1 = " + param1,
                "Param2 = " + param2,
                "Param3 = " + param3
               ];

    workspace.showInfoBox("ELO", content.join("<br>"));
    return 17
}
```

Aufgerufen wird das Skript dann über die COM Schnittstelle, z.B. über dieses kleine VBS Programm:

```
Set ELO = CreateObject("ELO.enterprise")
Result = ELO.RunScriptFunction("eloComMyScript", "Test1¶Test2¶Test3")
MsgBox Result
```

Wenn man statt eines Integer-Return Werts einen String benötigt, steht die Funktion RunScriptFunctionEx zur Verfügung.

```
function eloComMyScript(param1, param2, param3) {
    var content = [
                "<h3>MyScript</h3>",
                "Param1 = " + param1,
                "Param2 = " + param2,
                "Param3 = " + param3
               ];

    workspace.showInfoBox("ELO", content.join("<br>"));
    return "My Return Value"
}
```

```
Set ELO = CreateObject("ELO.enterprise")
Result = ELO.RunScriptFunctionEx("eloComMyScript", "Test1¶Test2¶Test3")
MsgBox Result
```

[10.2] VERWENDUNG DER OLE AUTOMATION EMULATION

Der Java Client besitzt eine Emulationsschnittstelle für das COM Interface des Windows Clients. Diese Emulation deckt ca. 70% der Funktionen ab. Manche Teile können aufgrund der Client Server Architektur gar nicht emuliert werden – z. B. die Events bei der Weiterleitung von Workflows. Da diese nicht im Java Client, sondern im Indexserver laufen, gibt es hier einen unüberwindlichen Unterschied zum Windows Client.

Manche Funktionen sind prinzipiell verfügbar, unterscheiden sich dann im Detail doch vom Original. Insbesondere, wenn man nicht dokumentierte Nebenwirkungen verwendet, kann man hier Differenzen beobachten.

Andere Funktionen weisen ein abweichendes Timing auf. Insbesondere die Reihenfolge von Events kann aufgrund der völlig unterschiedlichen Verarbeitung der Befehle erheblich abweichen. Ein Beispiel hierfür ist die Selektion eines Eintrags. Beim Windows Client wird der Inhalt des OLE Objekts bei jedem Klick auf einen Eintrag entsprechend aktualisiert. Das ist beim Java Client nicht möglich, da hier viele Aktionen asynchron auflaufen. Die Aktualisierung würde an manchen Stellen zu spät kommen. Oder sie würde einen nicht mehr aktiven Zwischenstand aufweisen. Aus diesem Grund gibt es einen zusätzlichen Befehl: UpdateSelection(). Diesen müssen Sie aufrufen, wenn Sie die Properties des aktuell selektierten Eintrags auslesen wollen.

DATEI IN DER POSTBOX KOPIEREN

Ein Kunde hatte sich im Windows Client angewöhnt, in der Postbox immer eine Art „Template Datei“ zu halten, aus der er bei Bedarf per ALT-Drag&Drop eine Kopie gezogen hat und dann diese verwendet hat.

Mal abgesehen davon, dass dieser spezielle Use Case etwas fragwürdig ist und besser über die Standardfunktion „Neues Dokument aus Vorlage“ abgebildet werden könnte – die Erzeugung einer Kopie der aktuell selektierten Postboxdokumente per Skript ist trivial:

```
function getScriptButton100Name() {
  return "Kopieren";
}

function getScriptButtonPositions() {
  return "100,intray,insert";
}

function eloScriptButton100Start(){
  var count = 0;
  var items = workspace.activeView.allSelected;
  while (items.hasMoreElements()) {
    var item = items.nextElement();
```

```
    var source = item.documentFile;
    intray.insertFile(source);
    count++;
  }
  workspace.setFeedbackMessage(count + " Dateien kopiert.");
}
```

VERSIONSABHÄNGIGE SKRIPTFUNKTIONEN

Die Java Client Skriptschnittstelle wird regelmäßig erweitert. Wenn man nicht sicher ist, ob beim Kunden eine neue Clientversion bereits verfügbar ist und die neue Funktion nicht unbedingt benötigt wird, kann man die Verwendung von der Client Versionsnummer abhängig machen.

```
var version = workspace.clientVersion;
if (version >= "9.03.000") {
  workspace.showInfoBox("neue Funktion", "die neue Funktion wird ausgeführt");
}
```

EXTERNE VIEWER EINBINDEN

Der Java Client erlaubt es nicht, dass Sie externe Viewer direkt einbinden können. Das wurde nicht implementiert, da instabile Viewer dann zum Absturz des kompletten Clients führen. Wenn Sie ein Dokumentenformat haben, welches der Java Client nicht unterstützt, haben Sie zwei Möglichkeiten:

1. die einfache Variante:

Falls es ein Browser PlugIn für dieses Format gibt, installieren Sie es und stellen im Java Client die Extension auf die Browser Anzeige um.

2. die umfangreiche Variante:

Die Version 9.2 erlaubt die Einbindung externer Viewer. Diese laufen als EXE Datei in einem eigenen Prozess und wenn sie abstürzen, hat das keinen direkten schädlichen Einfluss auf den ELO Client. Im Augenblick ist die Konfiguration noch manuell durchzuführen, das müssen wir noch komfortabler machen. Aber es funktioniert bereits.

A) Erstellen Sie in der Profileopts-Tabelle einen Eintrag mit dem Schlüssel "EloJ.S.ToPdf.<meine Extension>". In das optvalue Feld tragen Sie den Aufruf Ihrer EXE Datei mit dem Konvertierungsprogramm von <meine Extension> nach PDF. Das Programm bekommt zwei Parameter, die Sie als Platzhalter mit %1 für die Originaldatei und %2 für den Ziel-PDF-Dateinamen eintragen müssen.

EloJ.S.ToPdf.dwg "C:\Program Files (x86)\Any DWG to PDF Converter Pro\dp.exe" /InFile "%1" /OutFile "%2" /hide

B) Stellen Sie die Extension <meine Extension> im Java Client auf "PDF Vorschau" um.

Wenn nun eine Datei mit der Extension <meine Extension> geladen wird, ruft der Java Client vor dem Preview die konfigurierte EXE Datei auf. Diese konvertiert daraus ein PDF Dokument, welches dann vom Java Client zur Anzeige verwendet wird.

DATEIAUSWAHL-DIALOG

In Skripten benötigt man manchmal eine lokale Datei, die vom Benutzer ausgewählt werden muss. Dafür gibt es im workspace Objekt eine Methode zum Zugriff auf den Dateiauswahl-Dialog. Die Verwendung ist recht einfach. Als Parameter gibt man den Fenstertitel an, die Information, ob der Dialog zum Öffnen/ Lesen oder zum Speichern/ Schreiben von Dateien verwendet werden soll, ob nur Dateien oder auch Ordner ausgewählt werden können und der default-Pfad.

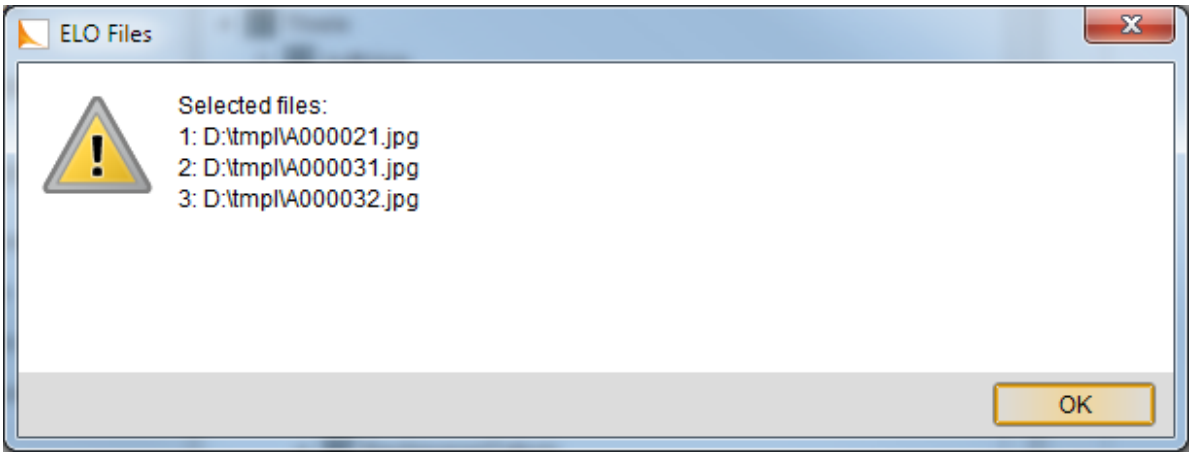
Das Skript ruft den Dialog auf und speichert den Pfad der ersten Datei als default-Pfad für die nachfolgenden Aufrufe. Anschließend werden alle Dateinamen aufgelistet und angezeigt.

```
var rememberPath = null;

function eloScriptButton100Start(){
  var files = workspace.showFileChooserDialog("Select file", false, true, rememberPath);
  if (files && (files.length > 0)) {
    rememberPath = files[0].path;

    var msg = "Selected files:<br>";
    for (var i = 0; i < files.length; i++) {
      msg += (i + 1) + ": " + files[i].path + "<br>";
    }

    workspace.showAlertBox("ELO Files", msg);
  }
}
```



EML ODER MSG MAILS AUSWERTEN

Im Java Client kommt man per Skript leicht an eine MSG oder EML Datei einer Mail heran. Wenn man aber Daten daraus auswerten möchte, ist das aufgrund der inneren Struktur dieser Dateien recht aufwändig. Aus diesem Grund gibt es unter utils eine Methode readMail(file). Diese liefert ein MailItem Objekt mit verschiedenen Informationen aus der Datei zurück.

```
function eloScriptButton100Start(){
  var item = workspace.activeView.firstSelected;
  var file = item.file;
  var mailItem = utils.readMail(file);
  workspace.showInfoBox("Subject", mailItem.subject);
}
```

Dieses Objekt eignet sich auch zum Auslesen der Attachments, Body-Text, Absender und Empfänger sowie Mail-Datum.

INTERVALLGESTEUERTE AKTIONEN AUSFÜHREN

Wenn man regelmäßig eine Aktion ausführen möchte, startet man in Java dafür einen eigenen Thread und lässt ihn zwischen zwei Aktivierungen die passende Zeit schlafen. Allerdings ist ein eigener Thread problematisch, wenn die Aktionen Änderungen an der GUI auslösen. Swing besteht darauf, dass solche Änderungen nur aus dem AWT-Thread ausgeführt werden – auch wenn nicht jeder andere Aufruf direkt zu einem Fehler führt.

Für einfache intervallgesteuerte Aktionen gibt es den Swing Timer. Er ruft regelmäßig eine Eventroutine aus dem AWT Thread heraus auf – allerdings nur, wenn gerade nichts passiert (was bei einer korrekt programmierten GUI der Normalfall ist). Und die Eventfunktion darf nur kurz laufen, sonst würde sie die GUI blockieren. Wenn diese Voraussetzung erfüllt ist, kann man diesen Fall sehr einfach abdecken:

```
function eloWorkspaceStarted() {
```

```
  globalTimer = new Packages.javax.swing.Timer( 2000, function(evt) {
    workspace.setFeedbackMessage(new Date());
  } );

  globalTimer.start();
}

function eloWorkspaceClosing() {
  log.info("Stop timer");
  globalTimer.stop();
}
```

In diesem einfachen Beispiel setzt die Eventfunktion nur die FeedbackMessage. Der Timer wird mit der Anzeige der Ansicht (eloWorkspaceStarted) gestartet und mit dem Schließen (eloWorkspaceClosing) wieder beendet.

HINTERGRUNDPROZESS STARTEN

Manchmal hat man langlaufende Auswertungen, die als Skript gestartet, den Client blockieren. Das ist für den Anwender ärgerlich, insbesondere da der Java Client viele andere langwierigen Aktionen in den Hintergrund verlagert.

Über die Methode workspace.startBackgroundProcess können Sie auch eigene Skripte im Hintergrund laufen lassen. Der Prozess wird auch in die Prozessliste des Java Clients eingetragen, so dass der Anwender erkennen kann, wenn die Arbeit abgeschlossen ist.

```
var process;

function eloScriptButton100Start(){
  process = workspace.startBackgroundProcess("sinus", "longRunningTask");
}

function longRunningTask() {
  try {
    var sum = 0.0;
    for (var i = 0; i < 100; i++) {
      for (var j = 0; j < 1000000; j++) {
        sum += Math.sin(i * j);
      }
```

```
      process.setStatus( i + " Prozent erledigt");
    }
  } finally {
    process.setFinished();
  }
}
```

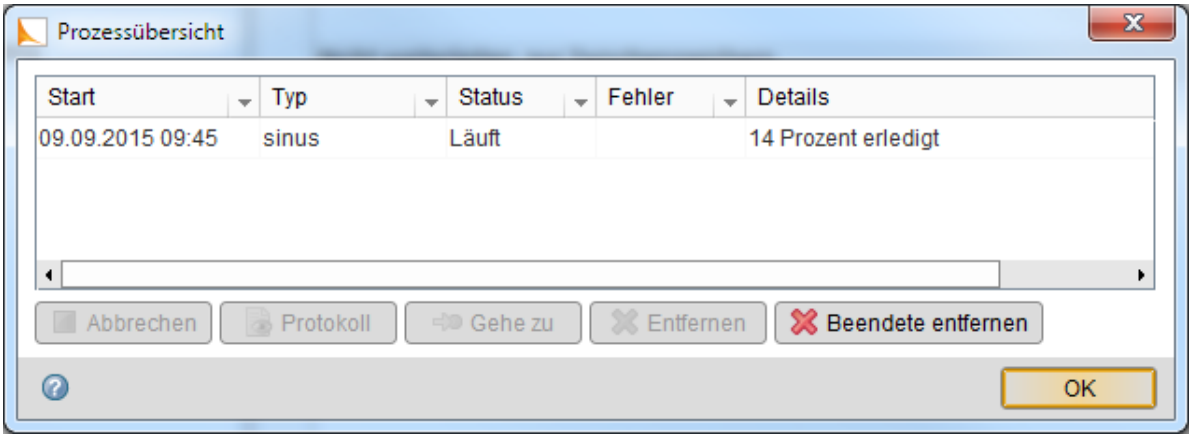


Abbildung 14: Statusanzeige für Hintergrundprozesse

Achten Sie unbedingt darauf, dass sich der Prozess am Ende selber aus der Liste der aktiven Prozesse austragen muss. Um das sicher zu stellen, wurde in der Skriptfunktion ein try – finally eingefügt, welches im finally Block process.setFinished() aufruft. Dieser Teil wird auch im Falle einer Exception ausgeführt und stellt sicher, dass keine beendeten Prozesse in der Liste stehen bleiben.

Über die Methode process.setStatus können Sie auch regelmäßig die Statusanzeige aktualisieren und somit dem Anwender eine Rückmeldung geben, wie weit der Prozess fortgeschritten ist. Achten Sie bitte darauf, dass der Status nicht zu oft aktualisiert wird, andernfalls ist der Client vorwiegend mit der Anzeige und weniger mit dem Prozessfortschritt beschäftigt. Diese Funktion sollte deshalb nicht häufiger als alle 5 Sekunden aufgerufen werden.

Achtung: Diese Funktion ist für Hintergrundprozesse vorgesehen. Im Wesentlichen können Sie nur die direkte Indexserverschnittstelle verwenden. Alle Befehle, die direkt das User Interface beeinflussen, sind hier unzulässig und können zu erheblichen Irritationen beim Anwender führen.

HINTERGRUNDPROZESS VORZEITIG BEENDEN

Wenn ein Prozess sehr lange dauert, wollen Sie dem Anwender auch die Möglichkeit geben, diesen vorzeitig abzubrechen. Da man einen laufenden Thread nicht einfach von außen abbrechen kann, muss sich das Skript selber darum kümmern. Dafür prüft es regelmäßig, ob das Stopped Flag gesetzt ist und wenn ja, beendet es sich selber.

```
var process;

function eloScriptButton100Start(){
  process = workspace.startBackgroundProcess("sinus", "longRunningTask");
}

function longRunningTask() {
  try {
    var sum = 0.0;
    for (var i = 0; i < 100; i++) {
      for (var j = 0; j < 1000000; j++) {
        sum += Math.sin(i * j);
      }
      if (process.isStopped()) {
        break;
      }

      process.setStatus( i + " Prozent erledigt");
    }
  } finally {
    process.setFinished();
  }
}
```

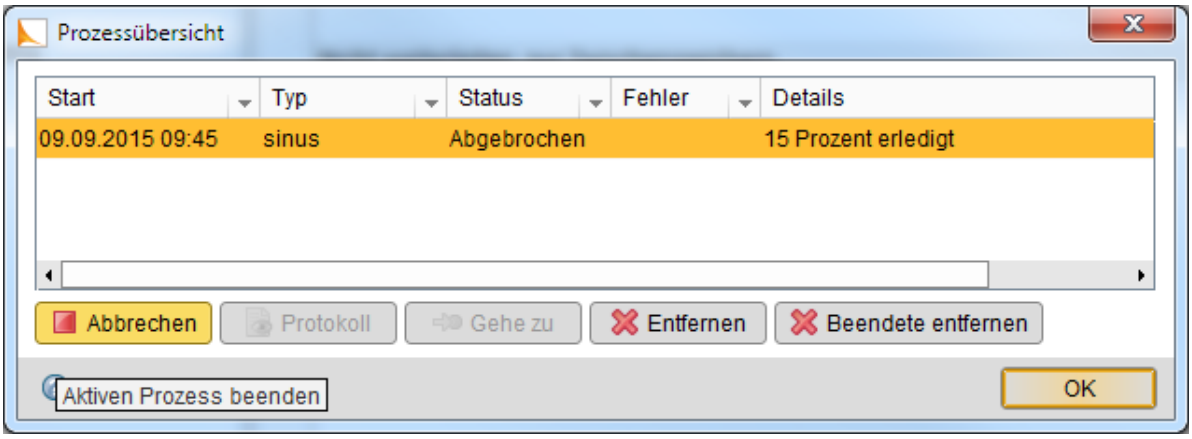


Abbildung 15: Prozess vorzeitig abbrechen

PROZESS MIT PROTOKOLLEINTRÄGEN

Damit ein Anwender leichter nachvollziehen kann, was in einem Hintergrundprozess abgelaufen ist, kann der Prozess ein Protokoll mitführen. Neue Einträge in das Protokoll können über die Methode process.addProtocolEntry eingefügt werden.

```
var process;

function eloScriptButton100Start(){
  process = workspace.startBackgroundProcess("sinus", "longRunningTask");
}

function longRunningTask() {
  try {
    var sum = 0.0;
    for (var i = 1; i <= 100; i++) {
      for (var j = 0; j < 1000000; j++) {
        sum += Math.sin(i * j);
      }
      if (process.isStopped()) {
        break;
      }
      var protMsg = "Sum of Sinus at " + i + "% is " + sum;
      process.addProtocolEntry( CONSTANTS.PROTOCOL_LEVEL.INFO, protMsg);
      process.setStatus( i + " Prozent erledigt");
    }
```

```
  } finally {
    process.setFinished();
  }
}
```

Es stehen die Protokoll-Level INFO, WARN und ERROR zur Verfügung. Der Anwender kann die Protokollanzeige gezielt nach einem Level filtern, um schneller bestimmte Einträge zu finden.

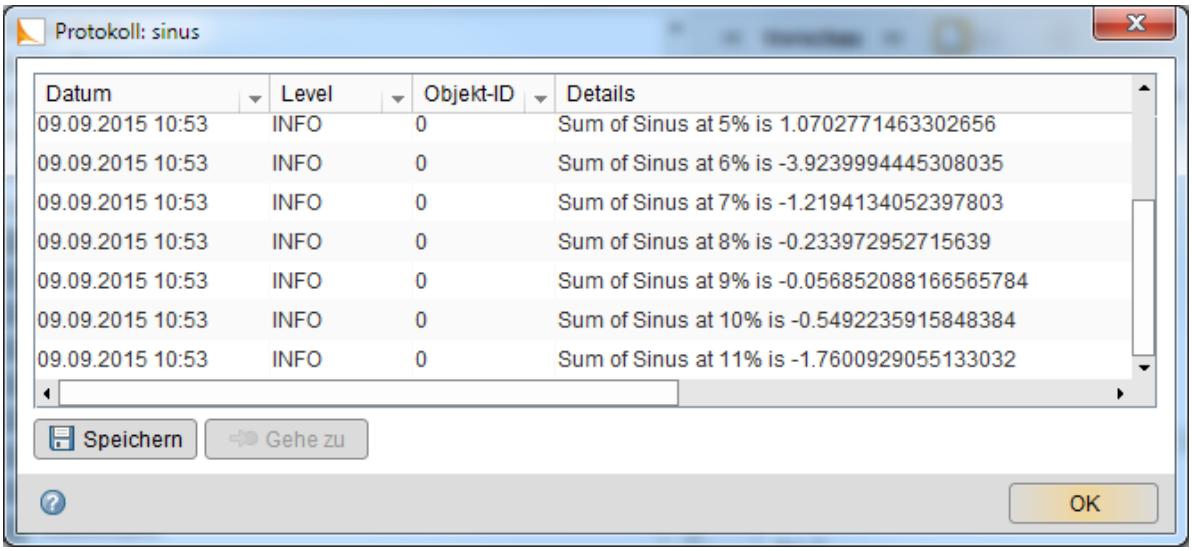


Abbildung 16: Protokoll aus der Skriptausführung

HINTERGRUNDPROZESS MIT FORTSCHRITTSANZEIGE

Ein Hintergrundprozess kann auch einen Fortschrittsbalken in einem Skriptdialog einblenden und aktualisieren. Hierfür bietet das GridPanel die Methode addProgressBar an. In Hintergrundprozess kann regelmäßig die Methode progressBar.setPercentage aufrufen, um die Anzeige zu aktualisieren.

Das Beispiel erzeugt einen Dialog mit einem Eingabefeld für die Anzahl der Berechnungsdurchläufe und einem Button für den Start der Berechnung. Die Aktionsroutine „startLongRunningTask“ des Buttons setzt den Dialogstatus auf „Berechnung aktiv“, blendet den ProgressBar ein und startet den Hintergrundprozess.

Der Hintergrundprozess aktualisiert regelmäßig den ProgressBar und die Ergebnisanzeige. Wenn die Berechnung abgeschlossen ist, wird der ProgressBar wieder ausgeblendet und der Dialogstatus auf Neutral gesetzt.

```
var process;
var dialog;
var progressBar;
var cntField;
```

```
var resultField;

function eloScriptButton100Start(){
  dlgWithProgressBar();
}

function dlgWithProgressBar() {
  dialog = workspace.createGridDialog( "Sinus III", 4, 7 );
  var panel = dialog.gridPanel;

  panel.addLabel( 1, 1, 1, "Durchläufe" );
  cntField = panel.addTextField( 2, 1, 2 );
  panel.addButton( 4, 1, 1, "Berechnung starten", "startLongRunningTask" );

  progressBar = panel.addProgressBar( 2, 2, 2 );
  progressBar.visible = false;

  panel.addLabel( 1, 4, 1, "Ergebnis" );
  resultField = panel.addTextField( 2, 4, 2);

  dialog.setStatusYellow( "Starten Sie die Berechnung" );
  dialog.show( "", "stopLongRunningTask" );
}

function startLongRunningTask() {
  dialog.setStatusRed( "Berechnung aktiv" );
  progressBar.visible = true;
  process = workspace.startBackgroundProcess("Sinus III", "longRunningTask");
}

function stopLongRunningTask() {
  if (process) {
    process.stop();
  }
}
```

```
function longRunningTask() {
  try {
    var loopCount = parseInt(cntField.text, 10);

    var sum = 0.0;
    for (var i = 1; i <= 100; i++) {
      for (var j = 0; j < loopCount; j++) {
        sum += Math.sin(i * j);
      }
      if (process.isStopped()) {
        break;
      }

      resultField.text = sum;
      progressBar.setPercentage( i );
      process.setStatus( i + " Prozent erledigt");
    }
  } finally {
    process.setFinished();
  }

  progressBar.visible = false;
  dialog.setStatusNormal( "" );
}
```

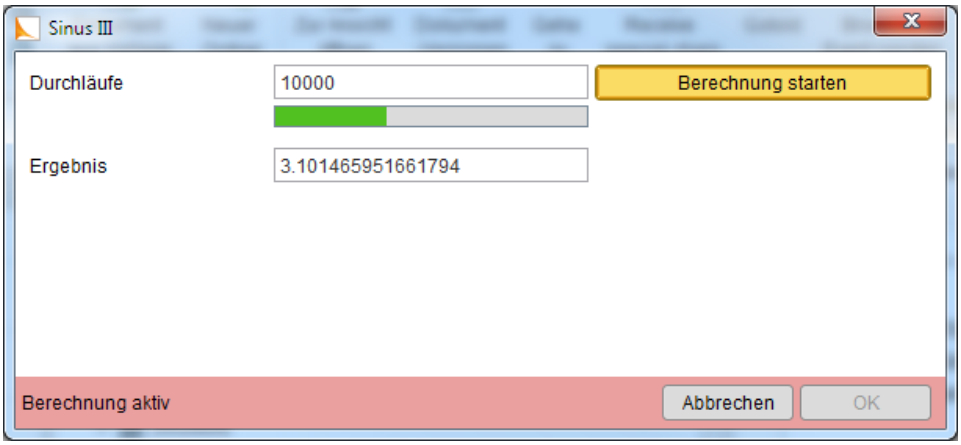


Abbildung 17:Laufende Berechnung mit Fortschrittsanzeige

KONFIGURATION FÜR SKRIPTE

Umfangreiche Skripte benötigen oft Konfigurationsdaten, z.B. über verwendete Masken, Archivpfade oder Workflows. Diese werden im schlimmsten Fall über das Skript verstreut – wo sie gerade jeweils benötigt werden. Etwas besser ist es, diese Daten als Konstanten am Anfang des Skripts einzutragen. Und noch besser ist es, ein einziges Konfigurationsobjekt am Anfang des Skriptes zu haben, damit es nicht so viele Einträge im Global-Namespace gibt (was bei umfangreichen Skripten aus unterschiedlichen Quellen zu Namenskollisionen führen kann). Allen Methoden gemein ist das Problem, dass Konfigurationsdaten, die auf die aktuelle Installation bezogen sind, mit den allgemeinen Skriptbefehlen in einer Quelle vermischt sind. Wenn es ein Skript-Update gibt, muss man darauf achten, dass die Konfigurationsdaten erneut angepasst werden.

Der Java Client hat deshalb einen Mechanismus, der es erlaubt, die Konfigurationsdaten komplett extern zu halten (z.B. in einem eigenen Ordner\Dokument: \Administration\Config\MyScriptConfig). Die Konfigurationsdaten im Dokument MyScriptConfig liegen im JSON Format vor und werden über einen @config Befehl zur Laufzeit in das Skript eingelesen.

```
// JavaScript Funktionen
//@config myObject /Administration/Config/Test Config
…
var objId = myObject.prop1;
```

MyScriptConfig enthält diesen Text:

```
{
  "prop1" : 4711,
  "prop2" : "Ein Test"
}
```

Zur Laufzeit im Debugger sieht das Skript dann so aus:

```
var myObject = JSON.parse('  {    \"prop1\" : 4711,    \"prop2\" : \"Ein Test\"  }    ');
```

Das @config Kommando enthält als ersten Parameter den Namen der Konfigurationsvariablen und als zweiten Parameter den Pfad auf das JSON Dokument im Archiv. Ziel ist es, dass alle ELO skriptfähigen Programme diese Konfiguration verstehen und einbetten können (ELOas, Indexserver, Java Client…).

HTML TEXT INS WINDOWS CLIPBOARD EINFÜGEN

Es ist relative einfach einen normalen Text ins Clipboard einzufügen. Etwas komplizierter wird es, wenn man einen HTML Text einfügen möchte. Unglücklicherweise gibt es hier noch Unterschiede zwischen Windows und dem Rest der Welt.

Das Windows HTLM Clipboard Format ist relativ komplex. Es gibt aber eine Hilfsfunktion in der Java Client Schnittstelle, die dabei behilflich ist. Man erzeugt nur den HTML Code und einen normalen Text (falls das Ziel kein HTML versteht) und ruft die Funktion addToClipboard auf.

```
function eloScriptButton100Start(){
  var sord = workspace.activeView.firstSelected.sord;
  var guid = sord.guid;
  var name = StringEscapeUtils.escapeHtml(sord.name);
  var htmlMsg = '<a href="elodms://' + guid + '">' + name + '</a>';
  var textMsg = 'elodms://' + guid;
  utils.addToClipboard(htmlMsg, textMsg);
}
```

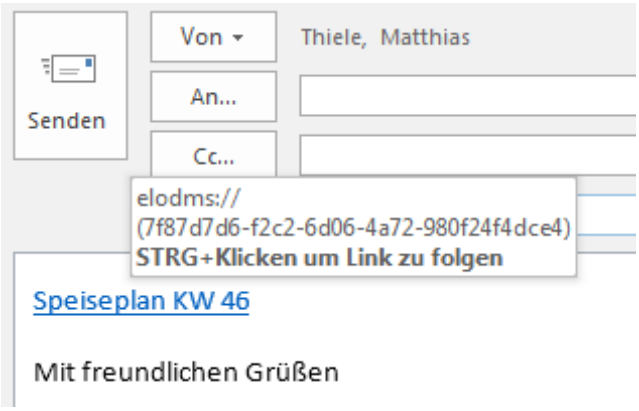


Abbildung 18: EMail mit elodms Link

URL Encoding ermitteln

Wenn man einen Text in eine URL einbetten möchte, gibt es eine Reihe von Sonderzeichen, die in der URL nicht enthalten sein dürfen und deshalb speziell codiert werden müssen (z.B. ? & / oder Leerzeichen).

Das folgende Skript fragt einen Text ab und zeigt die URL encoded Form an und trägt sie in das Windows Clipboard ein. Somit kann man den Text leicht in einen Link in einer Mail oder in Word übernehmen.

```
function eloScriptButton100Start() {
  var msg = "Geben Sie bitte den Text an, der in die URL eingebettet werden soll";
  var url = workspace.showInputBox("URL Encode", msg, "", -1, -1, false, -1);
  if (url) {
    var encoded = new Packages.java.net.URLEncoder.encode(url, "UTF-8");

    workspace.showInfoBox("Encoded", encoded);
    var clipboard = Packages.java.awt.Toolkit.getDefaultToolkit().getSystemClipboard();
    clipboard.setContents(new Packages.java.awt.datatransfer.StringSelection(encoded), null);
  }
}
```

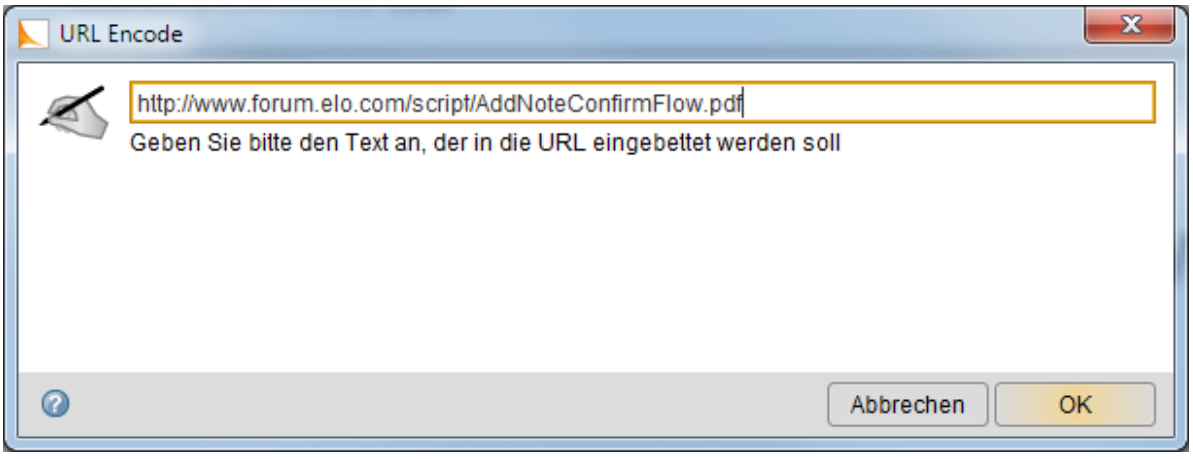


Abbildung 19: Eingabe des Quelltextes

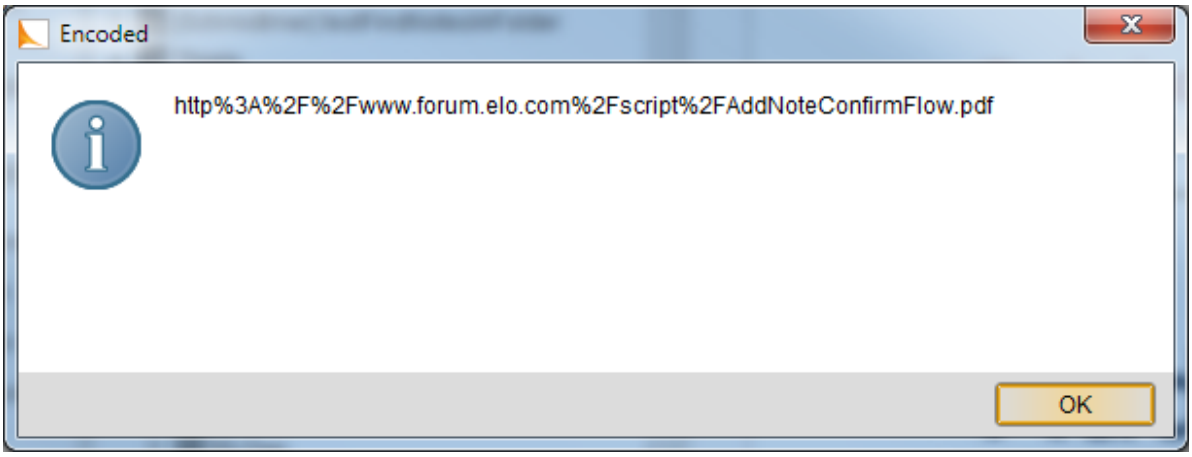


Abbildung 20: Ausgabe der codierten Form

[10] ZWEITE SEITE BEI DER ANZEIGE AUFBLÄTTERN

Manchmal hat man mehrseitige Dokumente mit einem Deckblatt. Das Deckblatt hat zwar notwendige Informationen, im Allgemeinen interessiert der Dokumenteninhalt aber mehr. Dann kann man über ein kleines Skript dafür sorgen, dass bei der Anzeige gleich auf die zweite Seite umgeschaltet wird.

Da die Anzeige asynchron läuft, ist das eigentlich nicht einfach. Allerdings gibt es im Client ein Cache, welcher in Information hält, welche Seite eines Dokuments zuletzt angesehen wurde. Er ist dafür da, dass ein Anwender, der mitten in einem Dokument steht und kurz auf ein anderes Dokument umschaltet, wieder auf die ursprüngliche Seite zurückgeführt werden kann. Diesen Cache kann man aber auch hervorragend für diese Aufgabe verwenden.

Das Skript wird in das Event bei der Anzeige eines Dokuments ein gehangen und prüft nach, ob der bestimmte Dokumententyp vorliegt – im Beispiel anhand des Maskennamens. Wenn ja, wird die Dokumenten-Id im Cache mit der zweiten Seite vorbelegt. Wenn dann später die tatsächliche Anzeige erfolgt, findet der Viewer diese Information vor und schaltet auf die gewünschte Seite um.

```
function eloPreviewAvailable(mode, item) {
  if (item && (item.docMaskName == "ZRechnung")) {
    workspace.setPageCachePage(item.id, 3, true);
  }
}
```

Der dritte Parameter in setPageCachePage entscheidet, ob der Eintrag immer vorgenommen wird (false) oder nur dann, wenn es noch keinen Eintrag gibt (true). In der aktuellen Form wird also ein Dokument, in dem der Anwender geblättert hat, nicht wieder auf die Seite zwei zurückgestellt. Wenn man prinzipiell auf die zweite Seite wechseln will, setzt man den Parameter auf false.

[10] HAFTNOTIZ IM DOKUMENT PER SKRIPT ERZEUGEN

Das Aufbringen einer Haftnotiz per Indexserver API ist eigentlich recht einfach, Beispiele hierzu finden sich in der IX Dokumentation. Das Hauptproblem bei Java Client Skript liegt darin, dass der Client Viewer erst mal nichts von der neuen Haftnotiz weiß. Auch ein refresh() reicht nicht aus, da dieses im Allgemeinen aus dem Client Cache kommt, der von den direkten IX Befehlen nichts mit bekommt. Zudem muss sichergestellt sein, dass noch nicht gespeicherte Haftnotizen des Anwenders vorher gesichert werden. Das passiert durch den Aufruf preview.deactivate() welcher die Änderungen der aktuellen Anzeige speichert und die Anzeige schließt.

Ab der Version 10 wird zusätzlich beim preview.reactivate der aktuelle Eintrag aus dem Cache gelöscht, bevor die Anzeige wiederhergestellt wird. Dadurch ist sichergestellt, dass die Anzeige komplett neu aus dem Indexserver geladen wird und somit auch die neu erzeugte Haftnotiz enthält. Die Version aus dem Cache würde diese nicht mit aufführen.

```
function eloHomeStart() {
  var item = workspace.activeView.firstSelected;

  var note = ixc.createNote2(item.id, NoteC.TYPE_ANNOTATION_NOTE_WITHFONT, "");
  note.noteText.text = "My note\n\nNote message with font";
  note.XPos = 100;
  note.YPos = 100;
  note.width = 800;
  note.height = 300;
  note.color = 0xffa040;
  note.noteText.fontInfo.faceName = "Arial";

  //preview.clear();
  preview.deactivate();

  ixc.checkinNotes(item.id, [note], NoteC.mbAll, LockC.NO);

  //workspace.updateSordLists();
  //workspace.activeView.refresh();
  preview.reactivate();

  return -1;
}
```

In älteren ELO Versionen kann man die Zeilen preview.deactivate() und preview.reactivate() entfernen und stattdessen die auskommentierten Zeilen preview.clear(), workspace.updateSordLists() sowie workspace.activeView.refresh() einfügen. Das funktioniert dann auch, löst aber sehr viel mehr Unruhe auf dem Bildschirm aus.

[10] ANZEIGE DER WANG ANNOTATIONS

Bevor wir die eigenen Annotations im Dokument hatten, haben wir die Wang Annotations unterstützt (so etwa um das Jahr 2000 herum). Nachdem Microsoft diese nicht weiter unterstützte, haben wir den Support dafür ebenfalls aufgegeben. Es gibt aber noch Anwender, die Dokumente mit den Wang Annoations im ELO gespeichert haben. Damit man diese weiter anzeigen kann, haben wir den Tiff-Viewer des Windows Clients auch als Kommandozeilenprogramm in das Java Client Setup aufgenommen. Der Viewer wird im Java Client Verzeichnis abgespeichert und kann durch einen einfachen Systemaufruf gestartet werden.

Das Beispielprogramm erwartet, dass ein Tiff-Dokument mit Wang Annotations selektiert ist. Wenn ein Anwender nun auf den Skriptbutton klickt, wird die (externe) Anzeige gestartet.

```
function getScriptButton100Name() {
  return "ELOmmView";
}

function getScriptButtonPositions() {
  return "100,home,view";
}

function eloScriptButton100Start() {
  try {
    var fileName = workspace.activeView.firstSelected.file.path;
    var cmd = ["C:\\Program Files (x86)\\ELO Java Client\\ELOmmView2.exe", fileName];
    var runtime = new Packages.java.lang.Runtime.getRuntime();
    runtime.exec(cmd);
  } catch(e) {
    workspace.setFeedbackMessage("Fehler bei der Anzeige " + e);
  }
}
```

Um den Anwender darauf hinzuweisen, dass das aktuell angezeigte Tiff Dokument Wang Annotations enthält, wird bei derartigen Dokumenten ein Warnhinweis „Nicht unterstützte TIFF Annotationen vorhanden“ eingeblendet.

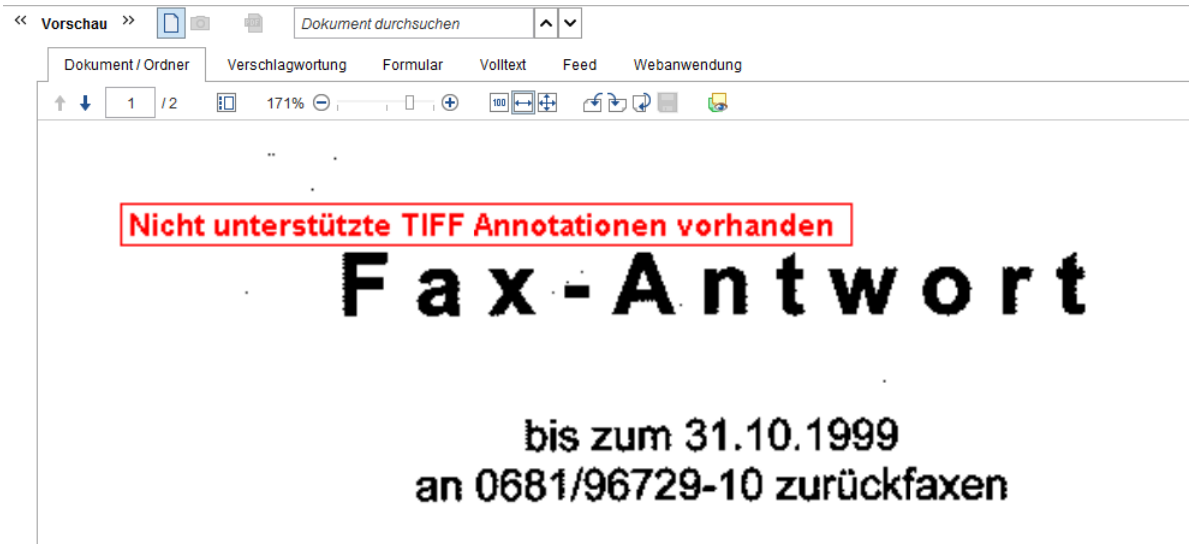


Abbildung 21: Warnhinweis bei einem Dokument mit Wang Annotationen

[10] AKTENZEICHEN GENERIEREN

Im Forum finden Sie ein komplettes Skript mit Beschreibung zur Implementierung eines Aktenzeichengenerators im Java Client.

Das Skript erlaubt eine weitgehende Konfiguration des Aktenzeichens über Indexzeileninformation, speziellen Eingabe- und Auswahlfeldern, festen Textanteilen und automatisch hochzählenden Countern.

```
parts : [
  { name : "OrgUnit", type : "edit", width : 6, regex : "^[1-3]+$", errmsg : "Nur Ziffern 1 bis 3 erlaubt" },
  { name : "Country", type : "combo", width : 10, data : ["Deutschland", "England", "Frankreich"] },
  { name : "Supervision", type : "check", width : 9, textTrue : "V/", textFalse : "" }
],
```

http://www.forum.elo.com/jforum/posts/list/3245.page#15482

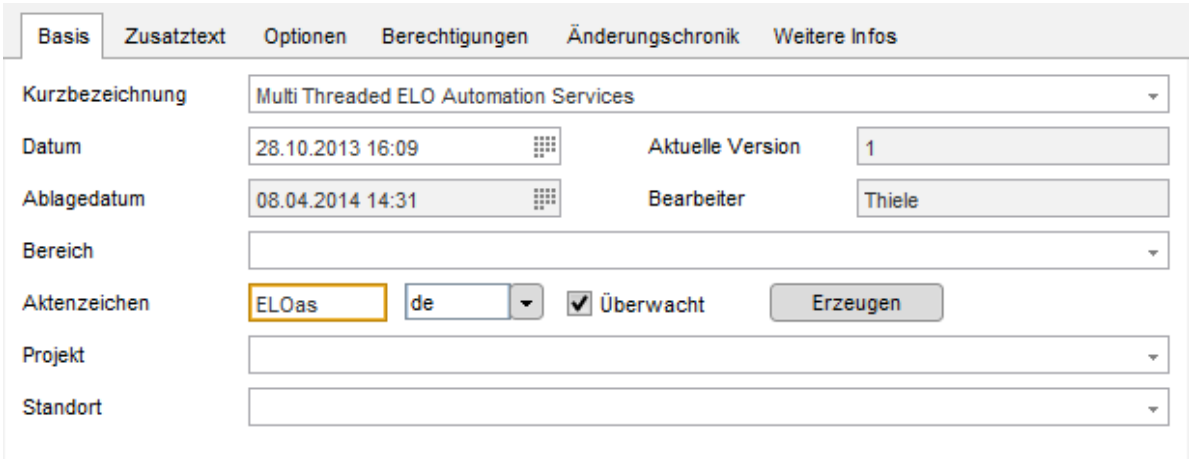


Abbildung 22: Verschlagwortungsmaske vor der Erzeugung des Aktenzeichens

Wenn der Anwender die nötigen Eingaben zur Generierung des Aktenzeichens durchgeführt hat und auf Erzeugen geklickt hat, werden die Eingabefelder ausgeblendet und stattdessen das Aktenzeichen angezeigt. Diese Indexzeile sollte in der Verschlagwortungsmaskendefinition als ReadOnly Feld deklariert werden, da es im Allgemeinen nicht erwünscht ist, dass ein vorhandenes Aktenzeichen nachträglich beliebig verändert werden kann.

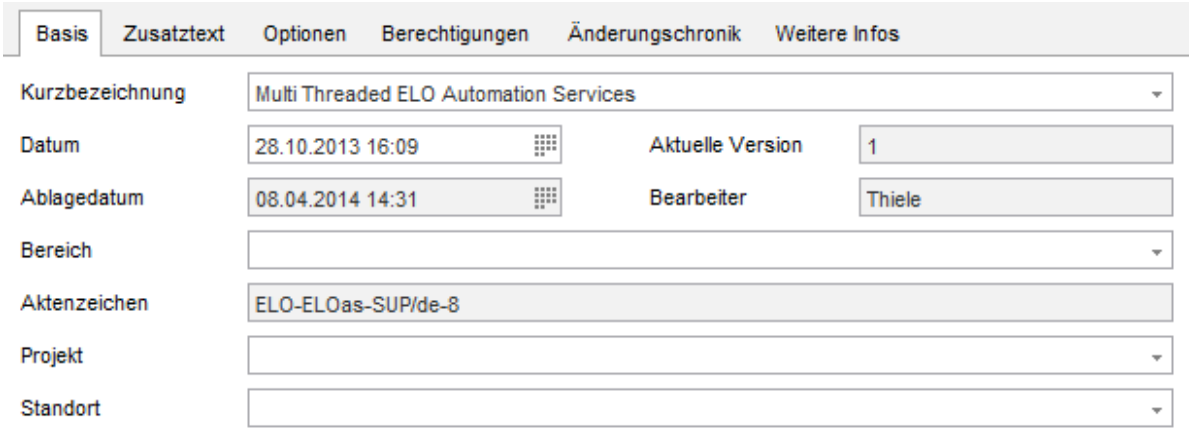


Abbildung 23: Verschlagwortungsanzeige mit Aktenzeichen

[10.1] ALLE EXIF TAGS EINER TIFF/JPEG DATEI ANZEIGEN

Man kann über die „Maskendefinition – Externe Daten“ bei der Verschlagwortung automatisch EXIF Tag Daten aus Tiff oder Jpeg Dateien in eine Indexzeile übernehmen. Hierzu muss man allerdings die vorhanden Tag-Namen kennen. Dieses Skript zeigt in der Postbox zum selektierten Dokument die vorhandenen Tags mit Namen und Wert an.

```
importPackage(Packages.de.elo.client);
importPackage(Packages.java.io);
```

```
function getScriptButton727Name() {
  return "Exif";
}

function getScriptButtonPositions() {
  return "727,intray,indexing";
}

function eloScriptButton727Start(){
  var item = workspace.activeView.firstSelected;
  var file = new File(item.filePath);

  var keys = utils.getExifTagNames(file);

  var buf = [];
  buf.push("<html><table>");

  while (keys.hasMoreElements()) {
    var item = keys.nextElement();
    var value = utils.getExifTag(file, item);
    buf.push("<tr><td>" + item + "</td><td>" + value + "</td>");
  }

  buf.push("</table></html>")
  workspace.showInfoBox("Exif", buf.join("\r\n") );
}
```

Neben den normalen EXIF Tags sehen noch ein paar Pseudo-Tags zur Verfügung:

FILENAME	Dateiname ohne Extension
FILENAMEEXT	Dateiname mit Extension
FILEDATE	Dateidatum
FILEDATEISO	Dateidatum im ELO ISO Format
FILESIZE	Dateigröße in Bytes
FILESIZEKB	Dateigröße in kBytes (/ 1024), aufgerundet
FILESIZEMB	Dateigröße in Mega-Bytes (/ (1024 * 1024)), aufgerundet

[10.1] Workflow-Annahme verhindern (Java Client)

Wenn man eine Vertretung einrichtet, kann der Vertreter alle Aufgaben der Person sehen und bei Bedarf annehmen und bearbeiten. Bei manchen Workflows ist das aber nicht sinnvoll. Ein Kunde hatte den Use Case, dass Weiterbildungen per Workflow abgearbeitet wurden. Es ist nun nicht passend, wenn ein Vertreter diese Aufgaben durchführt. Im Java Client kann man die Funktion AcceptWorkflow abfragen und prüfen, ob in der Liste der selektierten Einträge so ein spezieller Workflow enthalten ist. In diesem Fall kann man die Funktion abbrechen und eine Meldung ausgeben.

```
function eloAcceptWorkflowStart() {
  var items = workspace.activeView.allSelected;
  while (items.hasMoreElements()) {
    var item = items.nextElement();
    if (item.isWorkflow()) {
      var wfNode = item.task.wfNode;
      if (wfNode.flowName == "Weiterbildung (persönlich)") {
        workspace.setFeedbackMessage("Sie haben einen Workflow selektiert den Sie nicht
annehmen können: " + wfNode.nodeName);
        return -1;
      }
    }
  }
}
```

[10.1] Barcode – Handscanner

Es gibt eine Vielzahl preiswerte Handscanner zur Barcode-Erkennung. Diese können leicht ohne weitere Treiberinstallation angeschlossen werden, da sie eine Tastatur simulieren. Der erkannte Barcode wird also einfach als eine Folge von Tastendrücken gesendet.

Das ist einfach zu implementieren und zu installieren. Aber diese Vorgehensweise hat einen massiven Nachteil: man kann eine Barcode-Erkennung nicht von normalen Tastatureingaben unterscheiden. Und der Barcode geht in das Fenster, welches zu diesem Zeitpunkt gerade (zufällig) aktiv ist.

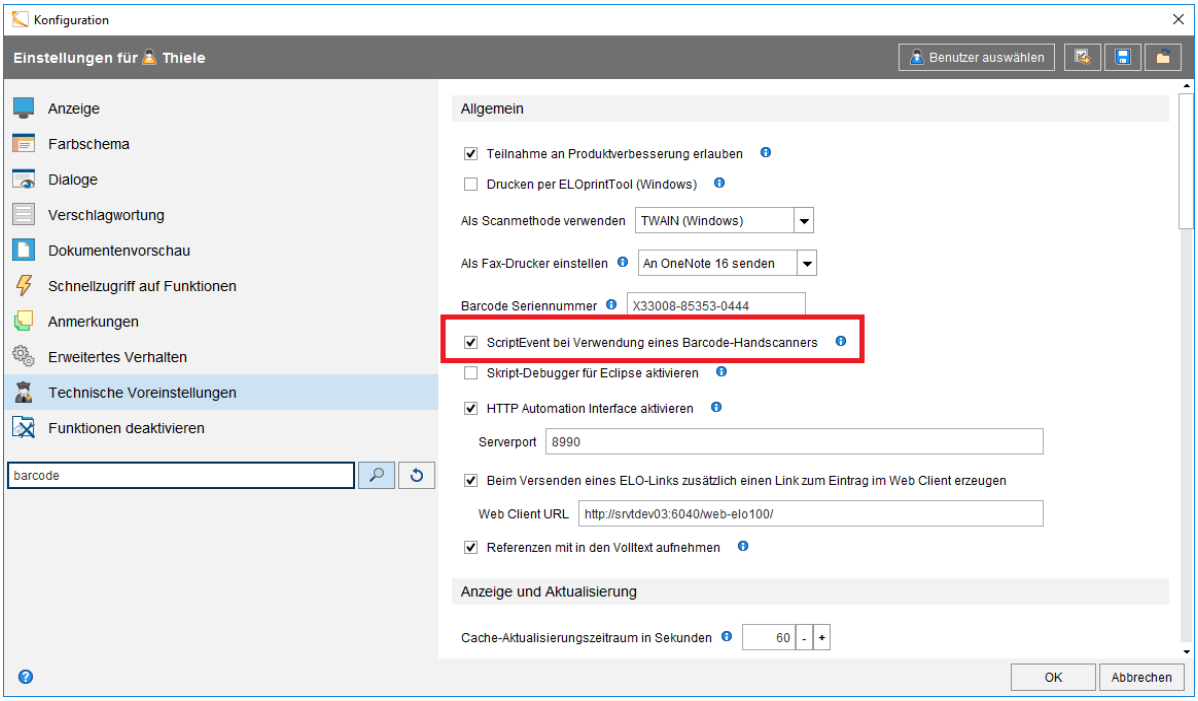
Wenn man bereit ist, diese Nachteile in Kauf zu nehmen, gibt es im Java Client zumindest eine kleine Hilfestellung bei der Barcode-Erkennung: man kann ein Skript-Event anmelden, welches dann gefeuert wird, wenn der Barcode Reader neue Daten gesendet hat. Wie kann der Java Client das erkennen – die Eingabe des Barcode Readers unterscheidet sich doch nicht von einer normalen Tastatureingabe? Es gibt doch einen Unterschied: die meisten Reader senden die Daten viel schneller, als ein Mensch sie eingeben könnte. Deshalb kann der Java Client die Tastaturschnittstelle überwachen. Immer dann, wenn

ein Block in hoher Geschwindigkeit hereingekommen ist, wird dieser an das Skript-Event gesendet und kann dort ausgewertet werden.

```
function eloBarcodeRecognized(barcode) {
  workspace.setFeedbackMessage(barcode);
}
```

Man muss dabei aber beachten, dass das nur funktioniert, wenn der Java Client das aktive Fenster besitzt (sonst werden die Daten gar nicht zum Java Client gesendet). Und, dass die Eingaben trotzdem an das aktive Fenster gehen. Wenn das Fenster keine Tastendrücke erwartet, werden sie einfach ignoriert. Wenn das Fenster aber in einem Eingabefeld steht, wird der Barcode dort eingetragen.

Da diese Barcode-Erkennung nicht 100% sicher läuft, muss sie über eine Konfigurationseinstellung freigeschaltet werden.



Man kann diese Funktion z.B. dafür nutzen, direkt eine Suche auszuführen. Wenn z.B. auf dem Lieferschein ein Barcode aufgedruckt ist und zu diesem Barcode Dokumente im ELO vorhanden sind (z.B. die Bestellung), dann kann man durch eine Handbewegung eine Suche im ELO auslösen. Analog zu dem Click&Find Beispiel wird eine automatische (Volltext)Suche ausgeführt, diesmal mit dem übergebenen Barcode aus dem Skript-Event. Und schon hat man alle Dokumente zu dem Lieferschein in der Anzeige.

```
function eloBarcodeRecognized(barcode) {
  workspace.setFeedbackMessage(barcode);
```

```
  var view = searchViews.getSearchView("Barcode");
  if (!view) {
    view = searchViews.addSearchView("Barcode");
  }

  var fbf = view.createFindByFulltext(barcode);
  var fi = view.createFindInfoByDate("");
  fi.findByFulltext = fbf;
  view.search(fi, 1000, true);
}
```

[10.1] Automatisches CheckIn über die Skript Schnittstelle

Seit der Version 10.0 besitzt der Java Client eine Funktion zum automatischen CheckIn von Dokumenten, welche über die ELO Vorlagenverwaltung erzeugt wurden. Ab der Version 10.1 steht diese Funktion auch über die Skriptschnittstelle zur Verfügung. Wenn Sie ein CheckoutDocument Objekt haben, können Sie dieses über die Methode markForAutoCheckin zum automatischen CheckIn anmelden. Sobald die Datei nicht mehr gesperrt ist, weil die dazugehörende Anwendung geschlossen wurde, wird dieses Dokument automatisch eingecheckt.

[10.1] Zoom-Faktor per Skript einstellen

Wenn man mit unterschiedlichen Dokumententypen arbeitet, welche stark unterschiedlich Zoom Einstellungen benötigen, kann es für den Anwender lästig werden, wenn er beim Dokumentenwechsel ständig die Zoom-Einstellungen ändern muss. Aus diesem Grund gibt es nun die Möglichkeit, dass die Zoom Einstellung auch per Skript angepasst werden kann.

Das folgende Skript geht davon aus, dass alle Dokumente, welche mit einem A beginnen, auf die volle Seitenbreite gebracht werden sollen (FitToWidth). Alle anderen Dokumente sollen komplett angezeigt werden (FitToPage).

```
function eloPreviewAvailable() {
  var item = workspace.activeView.firstSelected;
  if (item) {
    workspace.setFeedbackMessage(item.name);
    if (item.name.substring(0, 1) == "A") {
      preview.viewer.setFitWidth();
    } else {
      preview.viewer.setFitPage();
    }
  }
```

```
}
```

[10.1] Beispiel für einen einfachen Datei-Import

Dieses Skript ist aus einer Kundenanfrage entstanden. Es gibt ein Windows Verzeichnis mit Zeichnungen. Diese sollen auf Button-Klick in ein ELO Verzeichnis importiert werden. Falls es dort bereits eine Zeichnung mit dem Namen der Importdatei gibt, soll diese vorher in ein anderes Verzeichnis „ungültig“ verschoben werden.

Da das Skript viele Basistätigkeiten bei der Programmierung zeigt (Umgang mit externen Dateien, Einträge im ELO suchen, verschieden oder anlegen, Erzeugung eines eigenen Buttons), habe ich es hier aufgenommen.

```
function getScriptButton100Name() {
  return "Import starten";
}

function eloScriptButton100Start(){
  dirImport.start();
}

function getScriptButtonPositions() {
  return "100,home,edit";
}

var dirImport = {
  sourcePath : "D:\\temp\\import",
  eloPath : "/Importschrank/Zeichnungen",
  invalidPath : "/Importschrank/Zeichnungen-ungültig",
  documentMaskName : "Freie Eingabe",

  start : function() {
    this.getParents();
    this.processAllFiles();
  },

  // Ermittelt die Ordner im Archiv für die aktuellen
  // und ungültigen Dokumente.
  getParents : function() {
```

```
  this.eloPathFolder = archive.getElementByArcpath( this.eloPath );
  this.invalidPathFolder = archive.getElementByArcpath( this.invalidPath );
},

// Läuft über alle Dateien aus dem Windows Verzeichnis
processAllFiles() {
  var sourceDir = new Packages.java.io.File( this.sourcePath );

  // Listet alle Dateien aus dem Importverzeichnis auf
  var files = sourceDir.listFiles();
  for (var i = 0; i < files.length; i++) {
    var file = files[i];
    // Unterverzeichnisse ignorieren
    if (file.isFile()) {
      this.processOneFile( file );
    }
  }
},

// bearbeitet eine Datei, prüft, ob verschoben werden
// muss und importiert anschließend die neue Datei.
processOneFile : function( file ) {
  var name = file.name;
  this.moveInvalidFile(name);

  // importiert die Datei
  var sord = this.eloPathFolder.prepareDocument( this.documentMaskName );
  sord.name = name;
  this.eloPathFolder.addDocument( sord, file.path );

  // wenn der Import erfolgreich war, wird die Datei gelöscht.
  file["delete"]();
},

// Verschiebt die alte Datei - falls vorhanden.
```

```
  moveInvalidFile : function( name ) {
    var oldpath = this.eloPath + "/" + name;
    try {
      // prüft, ob ein Eintrag mit dem Namen vorhanden ist
      var oldDocument = archive.getElementByArcpath( oldpath );
    } catch(e) {
      log.debug("kein altes Dokument vorhanden, nichts zum Verschieben.");
      return;
    }

    // wenn es einen Eintrag gibt, dann wird er verschoben
    oldDocument.moveToFolder( this.invalidPathFolder, true );
  }
}
```

Man wundert sich vielleicht über die etwas merkwürde Schreibweise beim Löschen der Datei:

file["delete"]();

Das hängt damit zusammen, dass das Wort delete in JavaScript ein reserviertes Wort ist und nicht für Methodennamen verwendet werden kann. Da die Methode in dem Java Objekt aber nun mal so heißt, kann man nur indirekt über die Array-Property Schreibweise darauf zugreifen.

[10.2] VERSIONSNUMMERN PER SKRIPT GENERIEREN

Neue Versionsnummern werden vom Java Client so vorgeschlagen, dass die Nummer der alten Arbeitsversion eins hochgezählt wird. Wenn die Zahl einen oder mehrere Punkte enthält, wird der letzte Abschnitt hochgezählt. Aus der Arbeitsversion 10 wird also 11 und aus 10.1 wird 10.2. Diese Vorgehensweise ist in vielen Fällen hilfreich – manchmal hat ein Kunde aber sehr eigene Vorstellungen davon, wie der Vorschlagswert gebildet werden soll. Dafür gibt es nun ein neues Skript-Event, welches vor der Anzeige des Versions-Dialogs aufgerufen wird: eloBuildVersionNumber.

Das Skript-Event besitzt einen Parameter, ein VersionInfo Objekt. Dieses enthält Informationen über die Art des Aufrufs, den Versionstext der Arbeitsversion und den automatisch generierten neuen Versionstext sowie das aktuelle Verschlagwortungsobjekt – falls vorhanden. Das Skript kann den neuen Versionstext nach eigenen Regeln bilden und in das Info Objekt eintragen. Es wird dann als Vorschlagswert für den Versionsdialog verwendet.

```
function eloBuildVersionNumber( info ) {
  log.info(info.type);
  log.info(info.oldVersionNumber);
```

```
  log.info(info.newVersionNumber);
  if (info.sord) {
    log.info(info.sord.name);
    info.newVersionNumber = info.sord.id;
  }
}
```

[10.2] PDF VIEWER PER SKRIPT UMSCHALTEN

Der integrierte PDF Viewer hat den Vorteil, dass er die Annotations direkt im Dokument anzeigen kann. Zudem bietet er den Zugriff auf die Dokumenteninhalte über die Klick-OCR. Dies Möglichkeiten bietet der im Browser eingebundene Acrobat Reader nicht. Dafür ist es für viele Dokumenteninhalte schneller in der Anzeige. Und es gibt Dokumente, die der integrierte Viewer nicht oder nicht vollständig anzeigen kann. Der Anwender kann deshalb leicht zwischen den beiden Viewern umschalten.

Manchmal kann man aber auch per Skript erkennen, welcher Viewer voraussichtlich der bessere die diesen Eintrag sein wird. Hierzu gibt es eine neue Funktion setPdfPreference im PreviewAdapter. Denkbar wäre z.B., dass komplexe Dokumente vom Typ „Produktkatalog“ besser über den Acrobat Reader angezeigt werden und Rechnungen für die Klick-OCR besser mit dem ELO Viewer. Hierzu kann man die Maskennummer des angezeigten Eintrags auswerten.

```
function eloPreviewAvailable(mode, item) {
  var useBrowser = item.docMaskId == 0;
  preview.setPdfPreference(useBrowser);
}
```

[10.2] DATEINAMEN FÜR DRAG&DROP OPERATION VERÄNDERN

Je nach Einstellung verwendet der Java Client als Dateiname für Drag&Drop Operationen entweder den internen Namen aus dem Cache oder einen automatisch generierten Namen aus der Kurzbezeichnung.

Es besteht aber auch die Möglichkeit, einen eigenen Namen per Skript zu generieren. Es wäre denkbar, dass man die ELO Objekt-Id an den Namen anfügt, damit man die externe Datei später wieder dem Eintrag zuordnen kann. Oder man führt Buch über die herausgezogenen Dateien indem man einen Counterwert anfügt. Das hat auch den Vorteil, dass man die gleiche Datei mehrfach in ein Verzeichnis herausziehen kann.

```
function eloGenerateExternalFilename(sord, doc) {
  var counterValue = workspace.incCounter("DragAndDropCounter", 1000);
  return "ELO_" + counterValue + "_" + sord.name;
}
```



[10.2] CHECKOUT VERHALTEN ÄNDERN

Wenn man viele Dateien markiert und auscheckt, werden auch viele Programminstanzen zu den ausgecheckten Dokumenten geöffnet. Manchmal möchte man die ausgecheckten Dokumente gar nicht direkt bearbeiten, sondern erst mal nur sperren. Dieses Verhalten kann man relativ einfach per Skript erreichen.

Das Skript prüft nach, wie viele Einträge markiert sind. Wenn mehr als 100 Einträge selektiert sind, wird nur eine Fehlermeldung ausgegeben. Wenn nur ein Eintrag selektiert ist, wird die normale CheckOut Funktion mit Aufruf der Anwendung ausgeführt.

Wenn zwischen 2 und 100 Einträgen selektiert waren, werden die Dokumente zwar ausgecheckt, die Anwendung dazu aber nicht aufgerufen. Der Java Client wechselt nur in die CheckOut Ansicht und zeigt die neuen Arbeitsdokumente an.

```
function eloCheckOutStart() {
  var selectionCount = workspace.activeView.selectionCount;

  if (selectionCount > 100) {
    workspace.setFeedbackMessage(CONSTANTS.FEEDBACK_TYPE.WARNING, "Es
können maximal 100 Dokumente auf einmal ausgecheckt werden.");
    return -1;
  }

  if (selectionCount > 1) {
    var items = workspace.activeView.allSelected;
    while (items.hasMoreElements()) {
      var item = items.nextElement();
      if (item.isDocument()) {
        item.checkOut();
      }
    }
```

```
    workspace.showCheckoutView();
    workspace.setFeedbackMessage(selectionCount + " Einträge ausgecheckt.");
    return -1;
  }
}
```

[11.0] PRÜFEN, OB MAN MITGLIED EINER GRUPPE IST

Mithilfe der Methode workspace.isMemberOf(gruppe) kann man leicht prüfen, ob der aktuell angemeldete Anwender Mitglied einer Gruppe ist. Die Abfrage kann über den Gruppennamen oder über die Gruppen-Id erfolgen. Hierüber können Sie in einem Skript z.B. entscheiden, ob ein Anwender einen bestimmten Skript-Button eingeblendet bekommen soll oder nicht.

Diese Prüfung ist auch erfolgreich, wenn man nicht direkt Mitglied der Gruppe ist, sondern die Mitgliedschaft über eine „Gruppe in Gruppe“ erreicht.

[11.0] HTTP-GET AUFRUF IN EINEN STRING LADEN

Über den Aufruf workspace.getFromUrl(url) kann man einfach eine Webseite abfragen und das Ergebnis zur weiteren Verarbeitung in einen String schreiben. Im einfachsten Fall geben Sie eine direkte URL an. Das folgende Beispiel lädt die Impressum Seite des ELO Web-Auftritts ein und sucht dort nach der Umsatzsteuer-Id.

```
var elo = String(workspace.getFromUrl("https://www.elo.com/de-de/impressum.html"));
var id = elo.match(/(DE [0-9]{8,9})/i);
workspace.showInfoBox("ELO", id[0]);
```

Diese einfache Funktionalität lässt sich bereits mit der älteren Funktion workspace.sendHttpRequest erreichen. Die neue Funktion bietet eine Platzhalterfunktion für ELO interne Daten, z.B. für das aktuelle Ticket, den angemeldeten Anwender oder aktive Archiv. Wenn man im Aufruf den aktuellen Anwendernamen benötigt, kann man es so schreiben: http://server/app?name={username}.

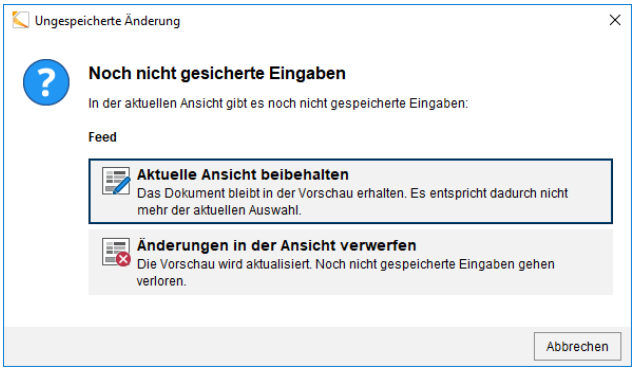
Folgende Platzhalter stehen zur Verfügung:

{archive}	Wird bei der Ausführung durch den aktuellen Archivnamen ersetzt.
{server}	Wird bei der Ausführung durch den Servernamen ersetzt.
{ticket}	Wird bei der Ausführung durch das aktuelle Session-Ticket ersetzt.
{userid}	Wird bei der Ausführung durch die Anwendernummer ersetzt.
{username}	Wird bei der Ausführung durch den Anwendernamen ersetzt.

Seien Sie vorsichtig, wenn Sie ein Session-Ticket an andere Dienste weiterreichen. Das Ticket ist wie ein Passwort. Wenn der fremde Dienst nicht vertrauenswürdig ist, kann er beliebige Aktionen in Ihrem Namen ausführen.

[12.0] NOCH NICHT GESPEICHERTE EINGABEN MELDEN

Wenn man im Java Client ein Formular bearbeitet und auf einen anderen Eintrag wechseln möchte, bekommt man einen Warnhinweis wenn es noch nicht gespeicherte Daten gibt, da durch den Wechsel das Formular neu geladen wird.



Wenn man diesen Warnhinweis auch in eigenen ELO Apps ausgeben lassen will, kann man über die Kommunikationsschnittstelle dem Java Client mitteilen, ob ein Wechsel eine Warnung auslösen soll oder nicht.

Hierfür gibt es die beiden Funktionen sendDirty() und sendSaved():

```
if (api.communication.Parent.hasParent()) {
    api.communication.Parent.sendDirty();
}
…
if (api.communication.Parent.hasParent()) {
    api.communication.Parent.sendSaved();
}
```

[12.0] EVENT BEI DER SELEKTIONSÄNDERUNG

Es gibt ein Event-Paar eloTreeSelectStart/End welches bei einer Selektionsänderung im Archivbaum aufgerufen wird. Dieses Event bekommt aber nicht mit, wenn die Selektionsänderung in der Ordnerliste stattfindet.

Ab der Version 12 gibt es deshalb ein zusätzliches Event eloSelectionChanged welches auch beim einem Klick in der Ordnerliste aufgerufen wird. Als Parameter wird das Java ListSelection Objekt übergeben. Dieses kann man z.B. dafür benutzen um die Zahl der selektierten Einträge zu ermitteln.

```
function eloSelectionChanged(event) {
  var selCount = event.source.selectedItemsCount;

  var item = workspace.activeView.firstSelected;
  if (item != null) {
    workspace.setFeedbackMessage(selCount + " : " + item.name);
  }
}
```

[12.0] EVENT AM ENDE EINES KOPIERVORGANGS

Wenn im ELO per Copy&Paste Verzeichnisstrukturen kopiert werden und im Anschluss daran noch Nacharbeiten per Skript durchgeführt werden sollen, gibt es ab ELO 12 ein neues Skript-Event eloPasteDone. Als Parameter bekommt man die Objekt-Id des Basisordners oder Dokuments im Zielbereich.

```
function eloPasteDone(objId) {
  workspace.setFeedbackMessage("New Item Root: " + objId);
}
```

[12.0] SELEKTIERTEN ARCHIVEINTRAG ERMITTELN

Die Skriptschnittstelle im Java Client ist außerordentlich symmetrisch was die unterschiedlichen Ansichten angeht. An vielen Stellen muss man noch nicht einmal wissen, in welcher Ansicht man sich befindet, weil viele Operationen in allen Ansichten verfügbar sind.

Beim Auslesen des aktiven Eintrags gibt es aber Unterschiede. Die Funktion firstSelected/ allSelected liefert in der Archivansicht direkt die Ordner/ Dokumentenobjekte. In der Aufgabenansicht aber Task-Objekte welche wiederum das Archivobjekt enthalten. Zudem liefern diese speziellen Listenansichten immer die Selektion auf der linken Seite - auch dann, wenn in der Ordnerliste rechts ein Eintrag selektiert ist.

Für die Fälle, in denen immer nur das Archivelement interessant ist, gibt es ab ELO 12 einen neuen Aufruf firstSelectedArchiveElement bzw. allSelectedArchiveElements. Diese liefern in allen Ansichten das selektierte Archivobjekt zurück und berücksichtigen auch eine eventuelle Selektion auf der rechten Seite.

Wenn man sich die Unterschiede mal ansehen möchte, kann man es mit diesem kleinen Skript machen:

```
function eloHomeStart() {
  var fs = workspace.activeView.firstSelected;
  var fsa = workspace.activeView.firstSelectedArchiveElement;
```

```
  workspace.setFeedbackMessage("fs: " + (fs ? fs.name : "null") + ", fsa: " + (fsa ? fsa.name :
"null"));
}
```

Eine Unterscheidung muss man nur in der Postbox machen: da es dort keine Archiveinträge gibt, liefert diese Funktion immer null zurück.

[12.0] SKRIPT-EVENTS IM MOBILE CONNECTOR

Ab der Version 12 gibt es zwei neue Skript-Events beim Start der Synchronisation und am Ende der Synchronisation über den Mobile Connector. Hier können Sie Vorbereitungen oder Nachbearbeitungen durchführen.

Die beiden Events heißen eloSyncStarted und eloSyncDone. Sie haben ein Parameterobjekt vom Typ SyncParams welches die Properties rootGuid (Startordner GUID im ELO Archiv) und rootFile (Startordner im Filesystem) enthält.

Im Start-Event können Sie zudem optional eine Callback Funktion anmelden, welche dann im Anschluss beim Lesen, Schreiben oder Löschen einer Datei oder eines Ordners aufgerufen wird. Sie können den Namen der Callback Funktion mittels workspace.setSyncCallback(<Name>) angeben. Dieser Aufruf muss in der eloSyncStarted Funktion durchgeführt werden, da der Callback vor jedem Start geleert wird.

```
var list;

function eloSyncStarted(param) {
  workspace.showInfoBox("ELO start", param.rootGuid + " --> " + param.rootFile.path);
  workspace.setSyncCallback("eloSyncItem");
  list = [];
}

function eloSyncDone(param) {
  workspace.showInfoBox("ELO done", list.join("<br>"));
}

function eloSyncItem(mode, sord, file) {
  var msg = mode + " : " + ((sord) ? sord.id : "-") + " : " + ((file) ? file.path : "-");
  list.push(msg);
}
```

Die Callback Funktion hat drei Parameter. Der erste enthält den Grund des Aufrufs, der zweite das aktuelle Sord Objekt und der dritte die aktuelle Datei oder Ordner. Die beiden letzten Parameter sind nicht immer vorhanden (bzw. können null enthalten). Das Sord Objekt ist nicht immer vollständig.

Dropzone und Java Client

SKRIPTKACHEL MIT OK/ FEHLERANIMATION

Eine Dropzone Kachel zeigt normalerweise bei der Ablage eine Ok oder Fehler-Animation an um den Anwender über den Erfolg der Aktion zu unterrichten. Diese Animation kann auch von einer Skriptkachel verwendet werden. Das Skript muss dazu einfach nur eine 0 für Ok oder eine -1 für Fehler zurückgeben.

```
function fromVBS_Animation(param) {
  var result = workspace.showQuestionBox("Dropzone", "Ja oder Nein");
  return (result) ? 0 : -1;
}
```

TRIM AUF EINEN GEFUNDENEN AUSDRUCK ANWENDEN

Ab der Version 9.03.002 wird die Dropzone bei ausgeschnittenen Begriffen und der Klammer mit dem Bereich einen zusätzlichen Teil für Modifier beinhalten. Dieser ist mit einem Pipe Symbol von den Positionszahlen abgetrennt.

Wenn also eine Datei „Rech Thm Stuttgart.txt“ lautet, gibt es nun folgende Ausgaben:

Beispiel (alt) – verwende aus dem Dateinamen ab der Position 4 die nächsten 5 Zeichen:

{File.NameNoExt(4,5)} ergibt „<Leerzeichen>Thm<Leerzeichen>“

Beispiel (neu) – gleichen Bereich, aber entferne vorne und hinten alle Leerzeichen.

{File.NameNoExt(4,5|T)} ergibt „Thm“

Beispiel (neu) – gleichen Bereich, ohne Leerzeichen und upper case.

{File.NameNoExt(4,5|TU)} ergibt „THM“

Im Augenblick gibt es diese drei Modifier:

T für Trim – entfernt führende und folgende Leerzeichen.

U für Upper – konvertiert in Großschreibweise.

L für Lower – konvertiert in Kleinschreibweise.

In der Zukunft kommen hier möglicherweise weitere hinzu.

Alle Modifier können beliebig kombiniert werden – auch wenn das bei U und L keinen Sinn ergibt.

MAILVERSAND ÜBER DIE DROPZONE

Die Dropzone selber kann keine Mail versenden, sie kann diese Aufgabe aber an ein Skript im Java Client übertragen. Hierzu wird in der Dropzone eine Skriptkachel „Mailversand“ eingerichtet. Neben Symbol und Farbe muss der Anwender nur noch den Skriptnamen JS.SendMail eintragen.

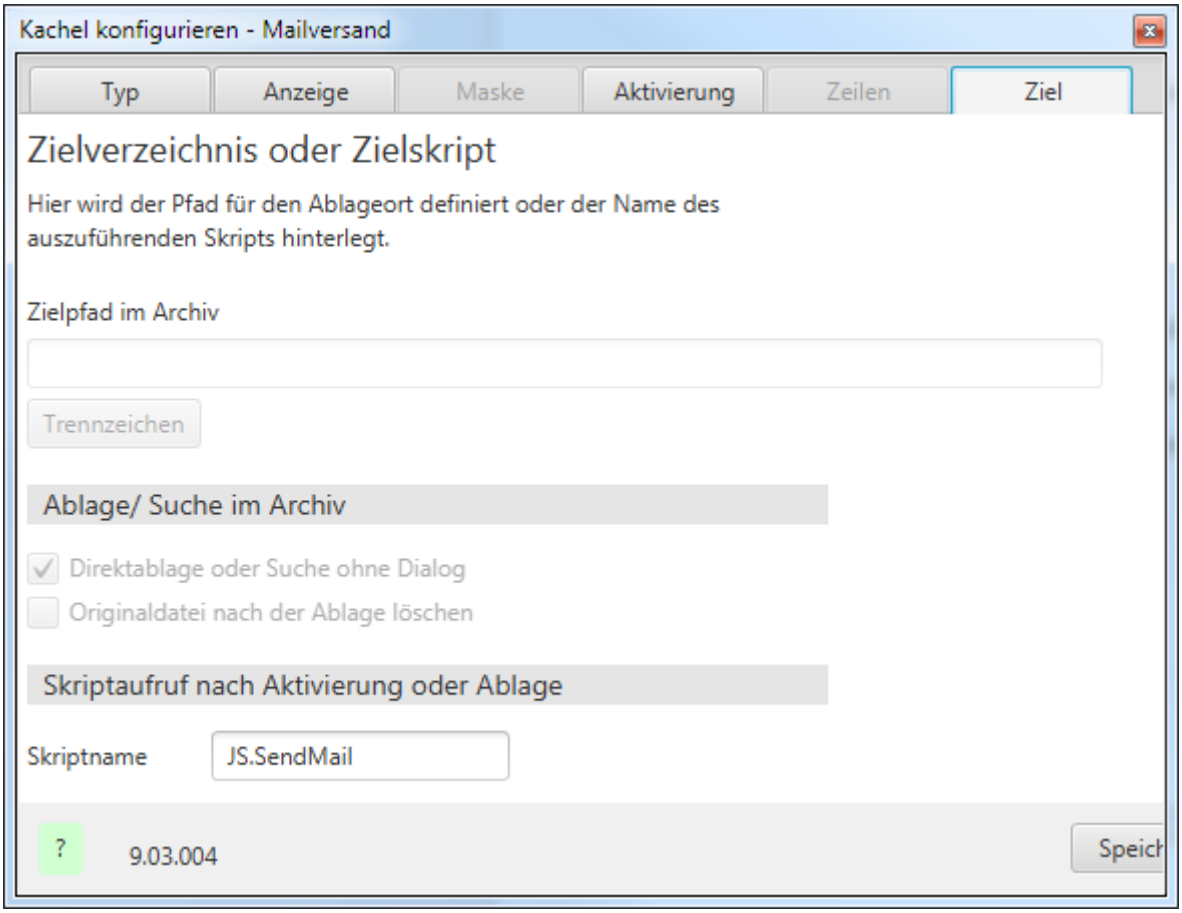


Abbildung 24: Konfiguration der Mailkachel

Das Skript fromVBS_SendMail im Java Client ist auch recht einfach. Die Dropzone übergibt alle ermittelten Werte in einem String-Parameter, dieser wird zuerst über die Methode fromString in eine Map aufgelöst. Aus dieser Map wird dann der Dateiname mit Pfad sowie der Dateiname ohne Extension ausgelesen. Diese Werte werden anschließend an die workspace Methode sendMail übergeben. Der Java Client kümmert sich dann um den eigentlichen Mailversand.

```
function fromVBS_SendMail(param) {
```

```
  var data = fromString(param);
  var fileName = data["File.Path"];
  var name = data["File.NameNoExt"];
  if (fileName) {
    workspace.sendMail(name, "", "", [fileName]);
    return;
  }
}

function fromString(param) {
  var result = new Object();

  var lines = param.split("\n");
  for (var i = 0; i < lines.length; i++) {
    var line = lines[i];
    var ipos = line.indexOf("\t");
    if (ipos > 0) {
      var key = line.substring(0, ipos);
      var value = line.substring(ipos + 1);
      result[key] = value;
    }
  }

  return result;
}
```

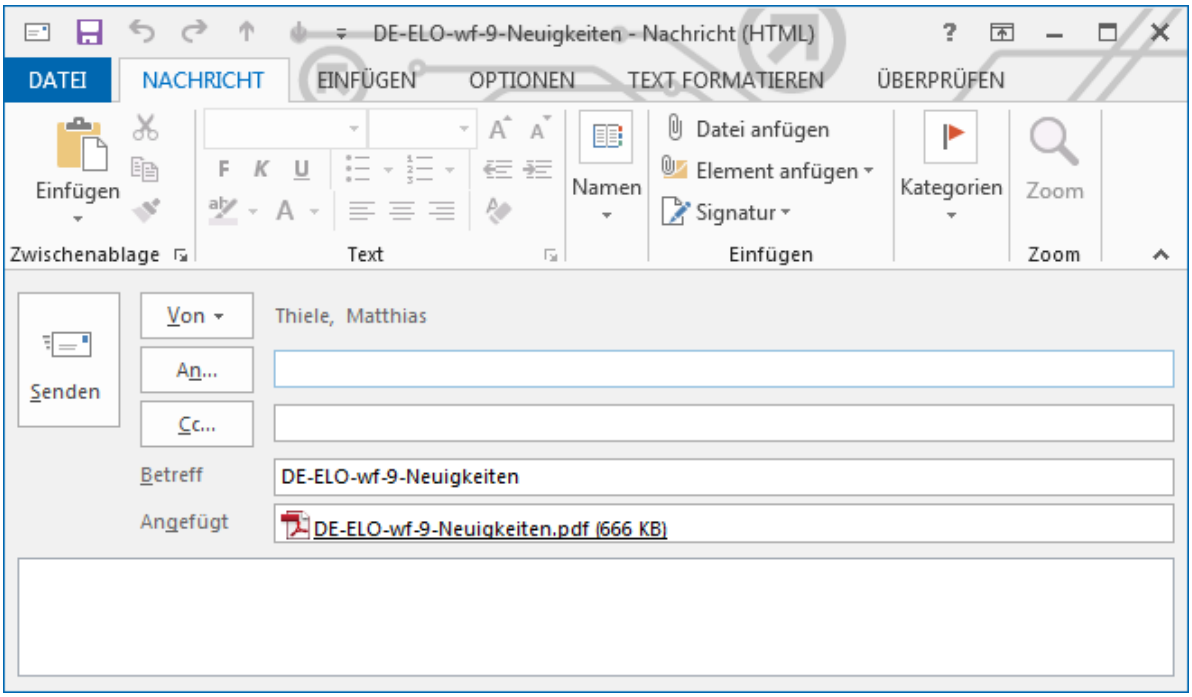


Abbildung 25: Zu versendende Mail im Outlook

TITLE-ERKENNUNG BEI MEHRDEUTIGEN EINTRÄGEN

Die Dropzone kann über reguläre Ausdrücke Informationen aus den vorhandenen Fenstertiteln gewinnen und zur Verschlagwortung verwenden. Das ist dann praktisch, wenn der Titel eindeutig auswertbare Informationen enthält. So kann z.B. ein Rechnungsprogramm die Kundennummer des aktuell bearbeiteten Kunden im Titel des Kundendialogs haben – z.B. in der Form „Kunde 4711 Maier“. Wenn nun eine Kundennummer zur Verschlagwortung benötigt wird, kann man einen regulären Titel-Ausdruck erzeugen, der auf den festen Text „Kunde “ (mit Leerzeichen) und einer anschließenden Zahl prüft. Diese Zahl wird dann als Kundennummer übernommen, der nachfolgende Text wird ignoriert.

Manchmal ist es aber komplizierter. Was ist, wenn es mehrere offene Kundendialoge gibt? Dann wird zufällig eine der beiden Kundennummern ausgewählt werden. Für diesen Fall gibt es die Möglichkeit des Eingabefensters oben in der Dropzone. Dort könnte man die Kundennummer manuell eintragen. Allerdings übernimmt dieses Eingabefeld auch immer die Daten aus dem aktuell aktiven Fenstertitel, wenn es einen passenden regulären Ausdruck gibt.

Als Beispiel soll der Notepad Editor verwendet werden. Er hält im Fenstertitel den aktuellen Dateinamen, gefolgt von dem festen Text „ – Editor“. Ein regulärer Ausdruck kann daraus den Dateinamen gewinnen: (.*) - Editor

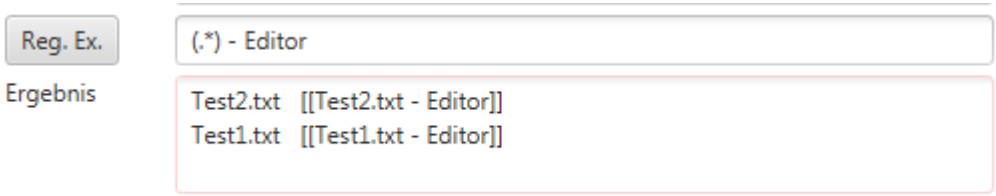


Abbildung 26: Zwei Notepad Versionen offen mit den Namen Test1 und Test2

Für sich alleine ist der Ausdruck nicht eindeutig, wenn mehrere Notepad Versionen offen sind. Wenn man nun aber den Notepad mit der Datei „Test1“ anklickt, wird dieser Wert in das Eingabefeld übernommen. Somit kann der Anwender bei Mehrdeutigkeiten zum einen genau erkennen, welcher Wert aktiv ist. Und zum anderen kann er den Wert durch eine passende Fensteraktivierung beeinflussen.

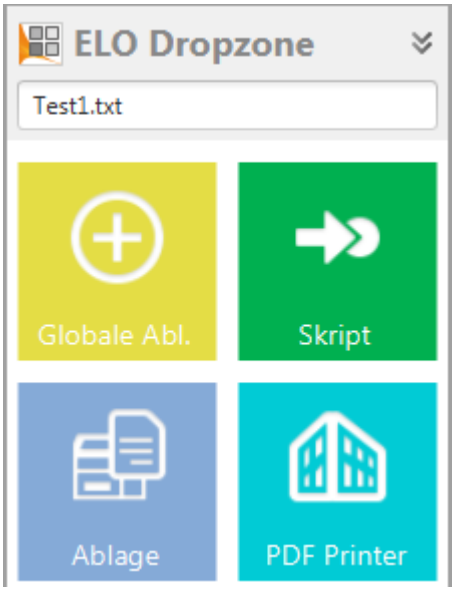


Abbildung 27: Datei Test1.txt ist aktiv

Aus der Werteliste für Indexzeilen kann dieser Wert aus „All.Manual“ ausgelesen werden.



Abbildung 28: All.Manual enthält den Wert des Eingabefelds

AUTOMATISCHE ABLAGE AUS DEM PDF DRUCKER

Die Dropzone kann über eine (oder mehrere) Kachel(n) das Ausgabeverzeichnis des PDF Druckers überwachen und neue Dokumente automatisch ablegen. Im Prinzip handelt es sich dabei um eine kleine Print&Archive Lösung.

Damit das Verzeichnis überwacht werden kann, muss zuerst das PDF Ausgabeverzeichnis in der Dropzone konfiguriert werden. Es muss nicht unbedingt der ELO PDF Printer sein, es kann eine beliebige Quelle verwendet werden.

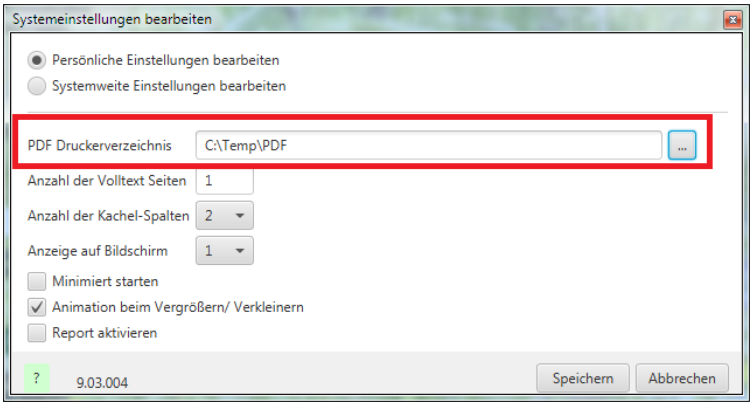


Abbildung 29: PDF Printer Verzeichnis eintragen

Jede Ablagekachel kann nun eine Überwachung dieses Verzeichnisses einrichten. Damit die Kachel erkennen kann, ob ein neues Dokument für sie gedacht ist, muss ein regulärer Ausdruck definiert werden, der im Volltext nach einem charakteristischen Text sucht. Wenn dieser Ausdruck einen Wert findet, ist die Kachel für das Dokument zuständig und legt es ab. Wird der Ausdruck nicht gefunden, dann ignoriert die Kachel diesen Eintrag.

TABELLEN IN REGULÄREN AUSDRÜCKEN

Eine wichtige Information bei der Bearbeitung von Word Tabellen über reguläre Ausdrücke: Jede Zelle erzeugt am Ende im Volltext ein unsichtbares \r. Das sieht man weder im Word Formular noch in der Volltextanzeige – muss aber beachtet werden, wenn sich ein regulärer Ausdruck über mehrere Zellen erstreckt.

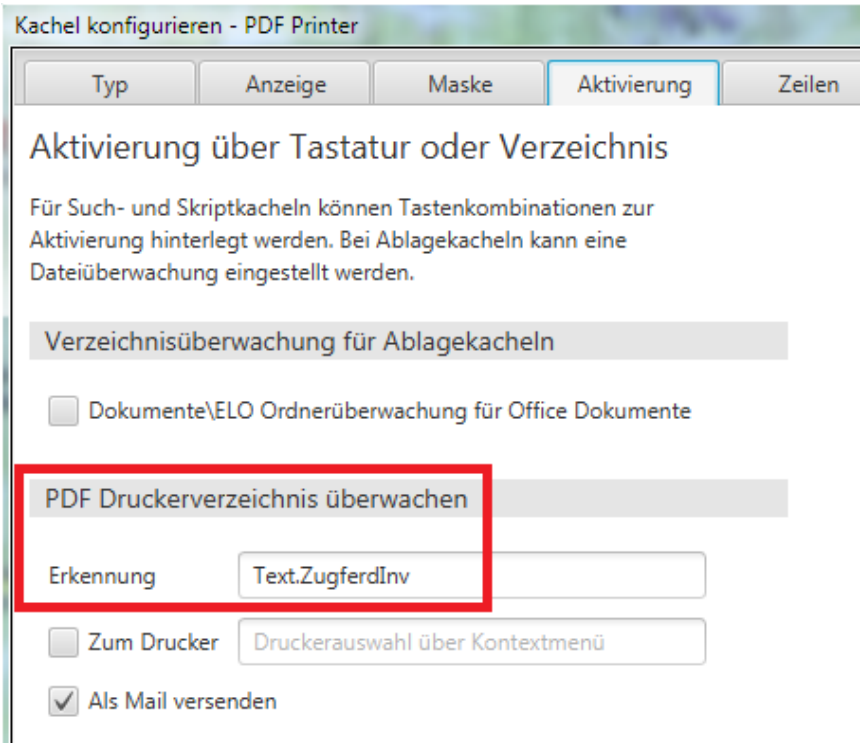


Abbildung 30: Selektion in der Kachel

Optional kann man über die Checkbox „Zum Drucker“ noch eine Ausgabe auf einen echten Drucker durchführen. Die Checkbox „Als Mail versenden“ erzeugt eine Mail mit der PDF Datei als Attachment.

[10] Dropzone Import über eine URL

Man kann ein Download und Import einer Datei aus einer Mail oder einer Web-Seite heraus in das Archiv auslösen. Dafür muss nur eine passend formatierte elodms URL hinterlegt werden. Der elodms Link beginnt in diesem Fall immer mit „elodms://dz/<Kachelname>|“.

Die URL kann entweder einen Dateinamen oder eine http Adresse beinhalten. Wenn eine Datei verwendet wird, dann muss diese lokal auf dem Client-Computer sichtbar sein. Der Dateiname wird dann URL-encoded an den elodms Link angefügt.

Im Normalfall wird man eine http Download Adresse angeben. In diesem Fall muss die Kennung „http“ angefügt werden, gefolgt von einem Pipe Symbol und der Extension (z.B. PDF). Daran schließt sich ein weiteres Pipe Symbol und anschließend die Download Adresse an. Beachten Sie auch hier, dass diese Teile URL-encoded geschrieben werden müssen, damit ein gültiger Link entsteht.

```
elodms://dz/Report|http|pdf|www.forum.elo.com%2Fscript%2FAddNoteConfirmFlow.pdf
```

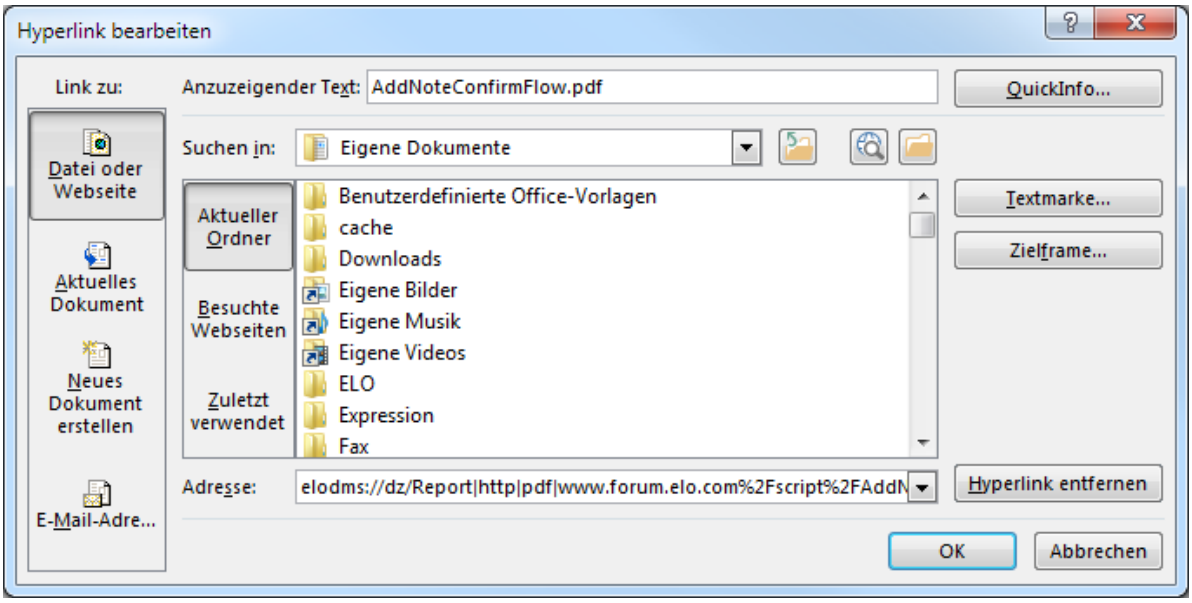
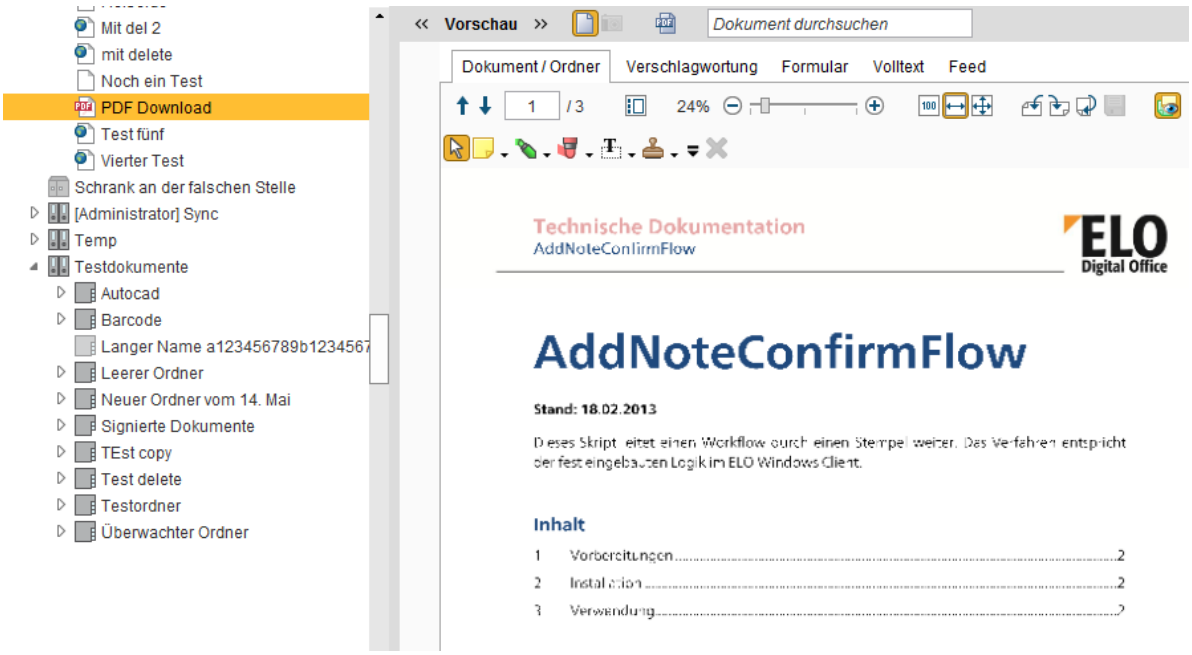


Abbildung 31: elodms Link in eine Outlook Mail einfügen

Wenn der Empfänger dieser Mail den Link aktiviert, wird die Datei heruntergeladen und an die genannte Kachel übergeben – so als hätte man sie per Drag&Drop auf die Kachel gezogen.



Wie sie eine URL in eine encoded Form überführen, können Sie in dem Kapitel „URL Encoding ermitteln“ nachlesen.

[10.2] BEREICHSSUCHE ÜBER DAS AKTUELLE JAHR ODER DEN AKTUELLEN MONAT

Wenn man eine Suchkachel auf Dokumente aus dem aktuellen Jahr einschränken möchte, kann man das über eine Bereichssuche durchführen. In diesem Fall gibt man in das Datumsfeld der Kacheldefinition den folgenden Wert ein:

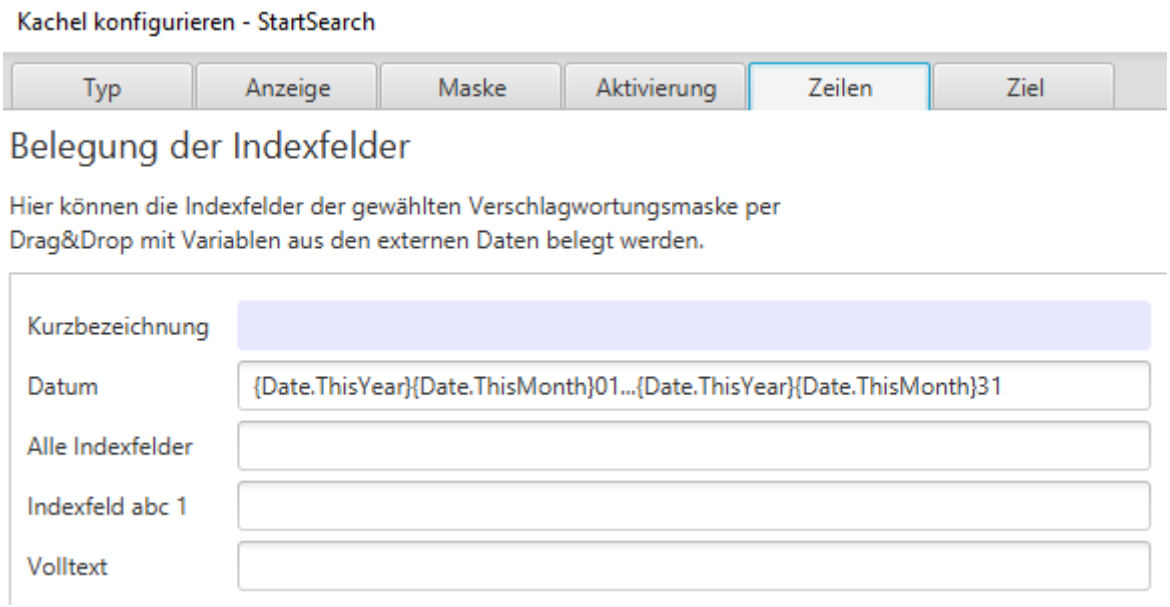
```
{Date.ThisYear}0101... {Date.ThisYear}1231
```

Date.ThisYear wird vor der Suche gegen die aktuelle Jahreszahl ausgetauscht und der Client führt eine Suche vom 1.1. bis zum 31.12. des aktuellen Jahrs aus.

Das gleiche gilt für eine Sucheinschränkung auf den aktuellen Monat:

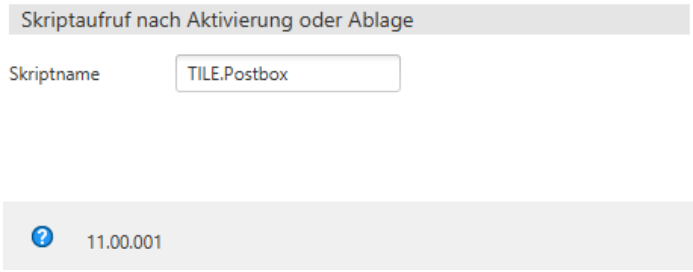
```
{Date.ThisYear}{Date.ThisMonth}01...{Date.ThisYear}{Date.ThisMonth}31
```

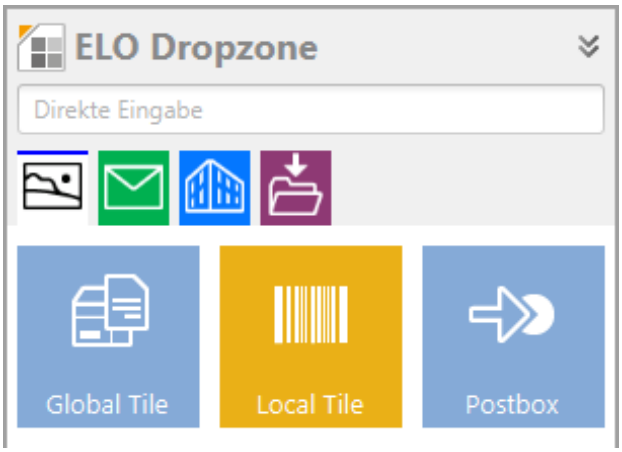
Hier gibt es aber eine Komplikation: nicht jeder Monat hat genau 31 Tage. Eine Suche im April 2017 erzeugt eine Bereichssuche 20170401...20170431. Einen 31. April gibt es nicht, das müsste eigentlich eine Fehlermeldung auslösen und gar keine Suche würde durchgeführt werden. Die Standard-Java Datumsroutinen sind hier aber etwas großzügiger – der 31. April wird kurzerhand zum 1. Mai (oder der 31. Februar 2017 wird zum 3. März). Die Suche wird also ausgeführt, bringt aber in den betroffenen Monaten manchmal zu viele Treffer. In vielen Situationen wird das aber keine Probleme verursachen – beim Dokumentendatum ist es eher selten, dass es Dokumente mit einem Datum in der Zukunft gibt.



[11.0] AUFRUF EINER JAVA CLIENT KACHEL ÜBER DIE DROPZONE

Manchmal wäre es praktisch, wenn man durch Klick auf eine Dropzone Kachel in einen Arbeitsbereich des Clients springen kann. Ab der Version 11 ist das möglich, indem man eine Skriptkachel anlegt und dort den Pseudo-Skriptnamen TILE.<Java Client Kachelname> hinterlegt.





Wenn man eine Kachel eines Such-Favoriten aufrufen möchte, muss das ausdrücklich mitgeteilt werden, da hier nicht nur die Ansicht umgeschaltet wird. Bei Favoriten muss die Ansicht erst erzeugt werden und nach der Umschaltung müssen die Favoriten-Einstellungen geladen und angezeigt werden. Deshalb sieht der Aufruf dann so aus:

TILE.FAVORITE.<MyFavoriteName>

[11.0] FAVORITEN-SUCHE ÜBER DIE DROPZONE

Man kann über die Dropzone nicht nur eine Favoritenansicht anzeigen (über TILE.<Favoritenname> im Skriptnamen), sondern auch eine Suche ausführen, welche den Suchbegriff aus den Dropzonedaten herausziehen kann. Hierzu legt man im Java Client ein Skript an, welches die Suche aktiviert und definiert dieses Skript anschließend als Kachel in der Dropzone.

```
function fromVBS_Reports(param) {
  log.info(param);
  var map = utils.splitParams(param);
  var search = map.get("NW.COMP");
  if (!search) {
```

```
    search = "";
  }

  var view = searchViews.getSearchView("favoriteSearch");
  if (!view) {
    view = searchViews.addSearchView("favoriteSearch");
  }

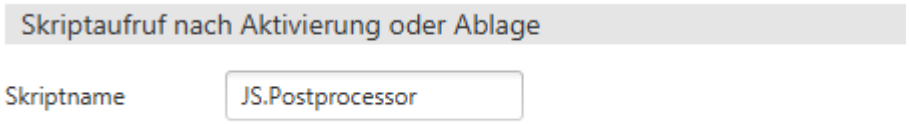
  view.triggerFavorite("Neue Reports", search, search != "");
  return -1;
}
```

Die Funktion utils.splitParams teilt den String, welches die Dropzone mit allen erkannten Werten sendet, in eine Map auf. Von hier aus kann man komfortabel auf jeden einzelnen Wert zugreifen.

[11.0] ANGELEGTE ORDNERSTRUKTUR BEARBEITEN

Wenn man die Ordnerstruktur durch die Ablage automatisch erzeugen lässt, hat man nur wenig Einfluss auf die Eigenschaften des erzeugten Ordners. Über ein Skript kann man diese bei Bedarf aber nachträglich ändern.

Ordner werden oft Alphabetisch sortiert, das ist in vielen Fällen auch sinnvoll. Wenn man aber eine mehrstufige Struktur hat und nur die letzte Ebene mit dem inversen Dokumentendatum sortiert haben will, kann man das nicht über die Maskendefinition erreichen. Hier kann man mit einem Skript, welches zu der Ablagekachel konfiguriert wird eingreifen.




```
function fromVBS_Postprocessor( param ) {
  var map = utils.splitParams(param);
  var id = map.get("LastStoredObjectId");

  var doc = archive.getElement(id);
  var parentFolder = doc.parent;
  var parentSord = parentFolder.sord;
  if (parentSord.details.sortOrder != SortOrderC.IXDATE) {
```

```
    parentSord.details.sortOrder = SortOrderC.IXDATE;
    parentFolder.sord = parentSord;
  }
}
```

Aus dem Skriptparameter LastStoredObjectId kann man das gespeicherte Dokument erreichen. Es wird anschließend der Parent Ordner dazu gelesen und geprüft, ob die Sortierung passend eingestellt ist. Wenn nicht, wird sie entsprechend geändert und der Ordnereintrag wieder gespeichert.

[12.0] RÜCKMELDUNG EINER SKRIPTKACHEL

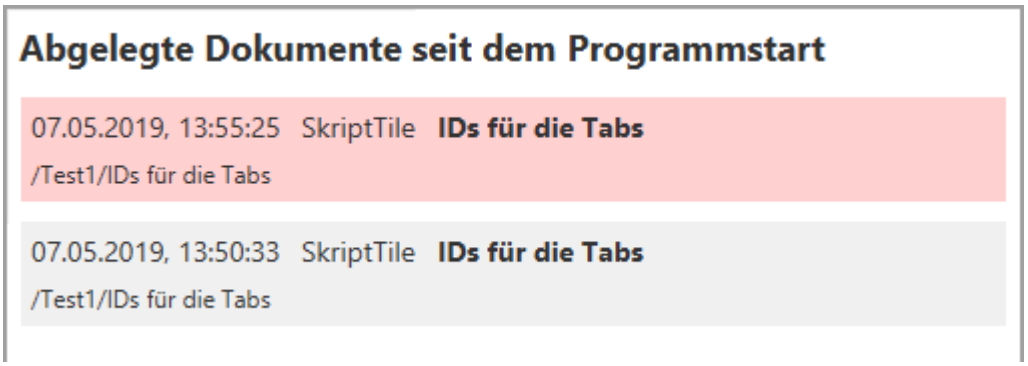
Wenn man eine Datei über eine normale Ablagekachel ablegt, wird dieser Vorgang in der Dropzone Übersicht gespeichert. Der Anwender kann hierüber auch nachträglich noch kontrollieren, ob der Vorgang erfolgreich war. Und er kann den neuen Eintrag durch Anklicken direkt anzeigen lassen.

Ab der Version 12 steht diese Möglichkeit auch Skriptkacheln zur Verfügung. Das Skript muss dazu den Anzeigetext und die neue Objekt-Id über ein Cookie JS.RESULT zurückmelden.

```
function fromVBS_Test(param) {
  var result = workspace.showQuestionBox("Dropzone", "Ja oder Nein");
  workspace.setCookie("JS.RESULT", "86706|SkriptTile|IDs für die Tabs|/Test1/IDs für die
Tabs");
  return (result) ? 0 : -1;
}
```

Der Cookie-Text enthält – jeweils durch ein Pipe-Symbol getrennt – die Objekt-Id, den Kachelnamen, den Titel und den Pfad für den Anzeigetext.

Der Rückgabewert des Skripts (0 oder -1) bestimmt, ob der Text als Ok oder als Fehler angezeigt wird.



Archivinformationen

ARCHIVPFADE EINES EINTRAGS ERMITTELN

Wenn man einen Ordner oder ein Dokument im Zugriff hat, ist es oft wichtig zu ermitteln, wo im Archiv dieser Eintrag liegt und ob es weitere Referenzpfade gibt. Diese Information liefert das Sord Objekt im Property refPaths.

```
function getScriptButton100Name() {
  return "Archivpfad";
}

function getScriptButtonPositions() {
  return "100,home,navigation";
}

function eloScriptButton100Start(){
  var firstSelElem = workspace.activeView.firstSelected;
  var elemPathArray = firstSelElem.sord.refPaths;
  var elemPathString = "";
  for ( var i=0; i<elemPathArray.length; i++ ){
     elemPathString = elemPathString + elemPathArray[i].pathAsString + "<br>";
  }

  workspace.showInfoBox( "Archivpfade:", elemPathString );
}
```

Dieses Property liefert ein Array von ArcPath Objekten. Der erste Eintrag ist der Ablageort im Archiv. Die optional zusätzlich vorhandenen Einträge sind zusätzliche Referenzen an andere Stellen im Archiv.

Wenn man nur den kompletten Pfad benötigt, bietet die Methode pathAsString des ArcPath Objekt einen direkten Zugriff. Benötigt man die Namen oder gar die Ids der einzelnen Ebenen, dann gibt im ArcPath Objekt das Property path. Es liefert ein Array von IdName Objekten welches für jede Ebene einen Eintrag besitzt – jeweils mit der Objekt-Id und dem Namen.

EINEN BESTIMMEN ARCHIVPFAD AUFBLÄTTERN

In einigen Installationen wird viel mit Referenzen gearbeitet um eine Struktur an verschiedenen Stellen anzuzeigen. Wenn man mit gotoId auf ein Dokument springt, wird immer der Hauptpfad verwendet.

Manchmal wäre es für den Prozess aber günstiger, wenn der Client eine Struktur über eine Referenz aufblättert.

Dafür gibt es die Methode gotoPath im ArchivView. Als Parameter geben Sie ein Array von IdName Objekten mit – hier tragen Sie für jede Ebene die gewünschte ELO Objekt-Id ein, die dort aufgeblättert werden soll. Diese Liste kann z.B. aus dem Sord Objekt gewonnen werden, wo unter refPaths alle Pfade, auch die über Referenzen, eingetragen werden.

Der Path kann auch aus der aktuellen Ansicht gewonnen werden. Dort kann man die Methode getFirstSelectedPath aufrufen. Er liefert den kompletten Pfad des selektierten Eintrags zurück. Dieser Pfad kann dann später dazu verwendet werden, genau wieder an diese Stelle zurückzukehren.

```
var rememberPath = null;


function eloScriptButton100Start(){
  if (!rememberPath) {
    rememberPath = workspace.activeView.firstSelectedPath;
  } else {
    workspace.activeView.gotoPath(rememberPath);
  }
}
```

Dieses einfach Skript merkt sich beim ersten Button-Klick die aktuelle Position (auch innerhalb eines dynamischen Registers) und kehrt bei jedem späteren Klick genau an diese Stelle zurück.

NUR LEERE ORDNER LÖSCHEN

Das Sord-Objekt stellt ein Property childCount zur Verfügung. Dieses eignet sich aber nicht dafür, um zu entscheiden, ob ein Ordner leer ist und deshalb gelöscht werden kann. Diese Vorgehensweise hat gleich zwei Fehler:

1. Das Property childCount enthält laut Dokumentation nur einen „geschätzten“ Wert. Das liegt daran, dass es sich bei dem Property um einen globalen Wert handelt. Der Anwender kann aber je nach Berechtigungen mehr oder weniger Dokumente sehen.
2. Selbst wenn der Wert in dem Moment des Auslesens frisch aus der Datenbank ermittelt werden würde, wäre er schon eine Millisekunde später fragwürdig. Denn in einer Millionen Nanosekunden können schon wieder neue Dokumente in den Ordner eingefügt worden sein.

Deshalb stellt der Indexserver beim Löschen eine Option „folderMustBeEmpty“ zur Verfügung.

```
var deleteOptions = new DeleteOptions();
deleteOptions.folderMustBeEmpty = true;
ixc.deleteSord(123, 456, LockC.NO, deleteOptions);
```

EINE BESTIMMTE SEITE IM DOKUMENT AUFBLÄTTERN

Über den Befehl workspace.gotoGuid kann man nicht nur einen bestimmten Eintrag anspringen – man kann auch gleich auf eine bestimmte Seite vorblättern. In diesem Fall muss die übergebene GUID mit der Seiteninformation ergänzt werden. Dafür wird ein @P und die Seitennummer am Ende angefügt.

```
workspace.gotoGuid( "(714466C2-7B76-715A-FD76-9D27C703EE7E)@P3" );
```

Alternativ dazu kann man auch eine bestimmte Annotation anspringen. In diesem Fall wird ein @A und die ELO Note-Id angefügt.

```
workspace.gotoGuid( "(714466C2-7B76-715A-FD76-9D27C703EE7E)@A12345" );
```

LISTE DER ANGEMELDETEN ANWENDER ERMITTELN

Streng genommen ist das keine Archiv-Aufgabe, sondern eine Aufgabe des AccessManagers. Da aber viele Installationen nur ein Archiv verwenden, wird das gerne synonym verwendet.

Eine Liste aller angemeldeten Benutzer kann man über die Indexserver Funktion checkoutUsers ermitteln. Dort gibt mal über den Parameter SESSION_USERS_RAW an, dass man nur die Liste der aktuell angemeldeten User haben möchte und nicht alle im System eingetragenen User.

```
getLoginCount: function () {
  var values = ixConnect.ix().checkoutUsers( null, CheckoutUsersC.SESSION_USERS_RAW,
LockC.NO );

  if (log.isDebugEnabled()) {
    for (var i = 0; i < values.length; i++) {
      log.debug("- " + i + " : " + values[i].id + " : " + values[i].name);
    }
  }

  return values.length;
}
```

PRÜFEN, OB EIN DOKUMENT VORHANDEN IST

Wenn man per Skript prüfen will, ob ein Dokument mit einem eindeutigen Verschlagwortungsbegriff bereits im Archiv vorhanden ist, muss man nicht unbedingt eine aufwändige Suche durchführen. Es geht in diesem Fall einfacher mit einem checkoutSord und der selten verwendeten OKEY: Schreibweise.

Aus der Indexserver-Dokumentation: Use "OKEY:<index-group-name>=<index-value>" to select an object by an index value, example: "OKEY:SAPPATH=123124109824123/data". The wildcards "%" or "*" can be used in <index-group-name> or <index-value> to return all or any data that matches.

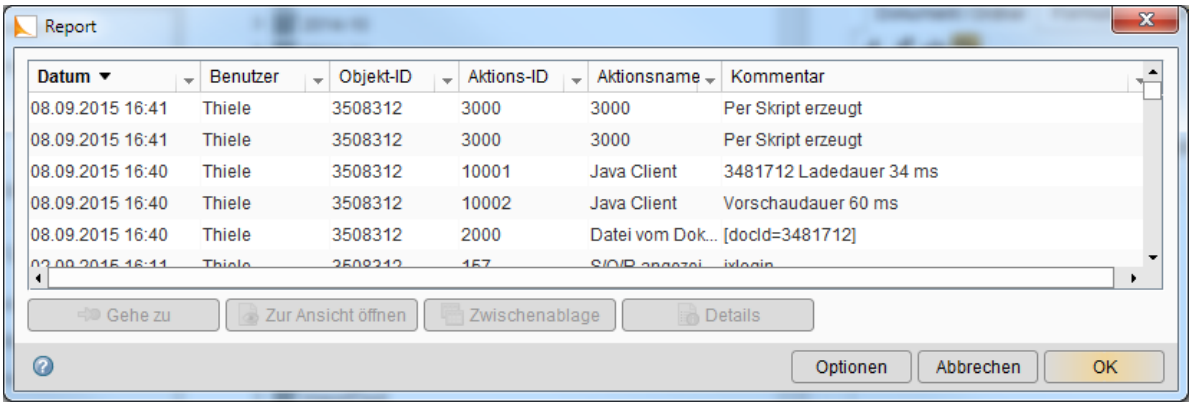
Voraussetzung für den Einsatz ist aber wirklich, dass der Begriff nur einmal im Archiv vorkommt. Andernfalls wird der Aufruf den Eintrag liefern, der zufällig als erstes in der Liste kommt.

CUSTOM-REPORTEINTRAG ERZEUGEN

In der Report-Tabelle werden alle wichtigen Ereignisse zu einem Ordner oder Dokument gespeichert. Per Skript können Sie dort auch Custom-Ereignisse eintragen. Diese haben eine ActionId im Bereich von 3000 bis 3999.

```
function eloScriptButton100Start(){
  var item = workspace.activeView.firstSelected;
  var id = item.sord.id;
  var actionNo = 3000;
  var extra1 = 100;
  var extra2 = 999;
  var message = "Per Skript erzeugt";
  workspace.addReportEntry(id, actionNo, extra1, extra2, message);
}
```

In der Report-Übersicht werden diese Einträge dann mit aufgelistet.



Leider werden in der aktuellen Version weder die Extra1 und Extra2 Felder mit ausgegeben noch ist es möglich, die Aktions-ID mit einem sprechenden Namen zu versehen. Das bleibt eine Aufgabe für spätere ELO Versionen.

BENUTZEREINSTELLUNGEN SPEICHERN

Wenn ein Anwender mehrfach hintereinander ähnliche Eingaben machen muss, ist es sinnvoll, diese zwischen zu speichern und beim nächsten Aufruf wieder vor zu belegen. Im einfachsten Fall geht das über eine globale Variable – wenn es mehrere Werte sind über ein Objekt in einer globalen Variablen.

Noch komfortabler für den Anwender ist es aber, wenn die Daten auch über eine Anmeldung hinweg erhalten bleiben. Hierfür kann man die Anwendereinstellungen in der profileopts Tabelle verwenden.

Da die Anwendereinstellungen Archivbezogen gespeichert werden, sind die Zugriffsfunktionen im archive Objekt hinterlegt: getUserOption und setUserOption.

Eine Optionseinstellung besteht im Wesentlichen aus zwei Teilen – dem Namen und einem Wert. Der Name sollte – um Kollisionen mit Namen aus anderen Skripten zu vermeiden – ein Kürzel des Firmennamens als Start enthalten. Als Beispiel: „contelo.S.exportDirectory“. Das S signalisiert eine String-Option (intern gibt es noch andere). Für ELO reserviert sind alle Optionseinträge, die mit ELO, Client, Mso oder WebClient beginnen.

```
var mode = 1;
var lastId = archive.getUserOption("EloJ.S.Script.ELOGotoId.LastId", "");

var result = this.inputIdDlg(lastId, mode);
if (!result) {
  return;
}

archive.setUserOption("EloJ.S.Script.ELOGotoId.LastId", result);
```

EXTERNE DATEIEN INS ARCHIV AUFNEHMEN

Bei Migrationsprojekten kommt es manchmal vor, dass es große Datenbestände auf einem WORM Device oder einer Disk Appliance mit einer Sperrfrist gibt. Wenn man diese Dokumente ins ELO importiert, wird der alte Speicherplatz nicht frei und am Ende hat man den doppelten Speicherbedarf.

Für diese Situation gibt es im Dokumentenmanger die Möglichkeit für externe Dokumentenpfade. In diesem Fall wird nicht eine Datei übergeben und neu abgelegt, sondern eine bereits vorhandene Dateireferenz gespeichert und der Zugriff erfolgt dann direkt auf die ursprüngliche Datei.

Um dieses Feature zu nutzen, muss der ELOdm natürlich Zugriff auf die Dateien haben und die Dateireferenz muss aus der Sicht des ELOdm übergeben werden. Zuerst wird in der Admin Console ein Pfad auf das Wurzelverzeichnis des Speichermediums angelegt.

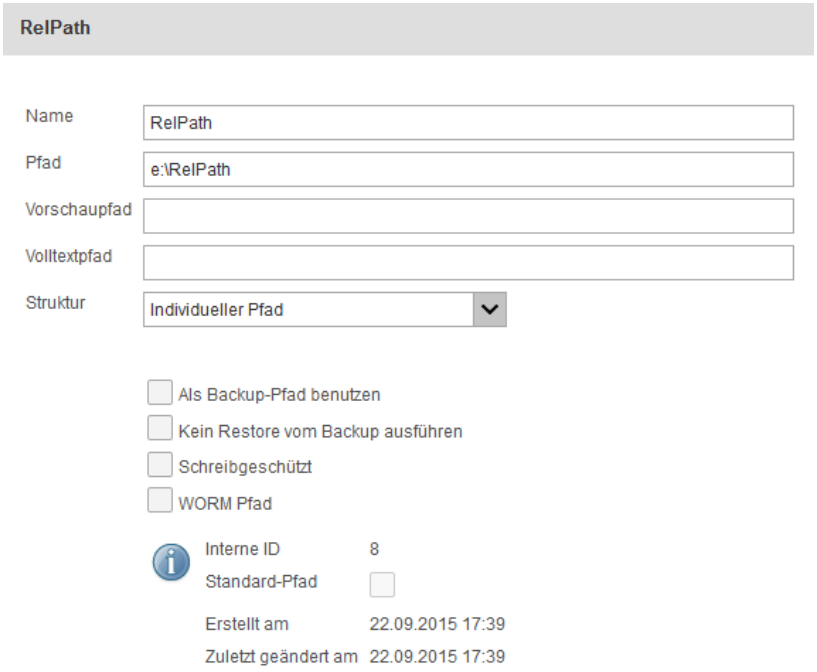


Abbildung 32: Individuellen Pfad für relative Dateien anlegen

Dieser Pfad kann nun für die Ablage der bereits vorhandenen Dokumente im ELO verwendet werden. Das kann z.B. über eine ELOas Regel oder ein Java Client Skript erfolgen, welches die Migrationsdatenbank ausliest, daraus die Verschlagwortung und den Dateipfad ermittelt und diese Daten dann an den Indexserver übergibt.

```
var parentId = workspace.activeView.firstSelected.id;

var ed = ixc.createDoc(parentId, "", "", EditInfoC.mbSordDoc);
var sord = ed.sord;
sord.name = "RelPath Document Zugferd Rechnung";
var dv = new DocVersion();
dv.pathId = 8;
dv.relativeFilePath = "folder1\\ZugInv.pdf";

var doc = ed.document;
doc.docs = [dv];

ixc.checkinDocEnd(sord, SordC.mbAll, doc, LockC.NO);
```

Die verwendete PathId (dv.pathId = 8) finden Sie in der Admin Console als “Interne ID”. Der komplette Dateipfad ergibt sich aus dem Wurzelverzeichnis der Pfaddefinition (e:\RelPath) und dem relativeFilePath aus der Dokumentenablage (folder1\ZugInv.pdf) -> e:\RelPath\folder1\ZugInv.pdf

[10] ANZEIGELISTEN NACH DEM SAMMELN BEARBEITEN

Über das Skript-Event eloCollectListStart kann man die Verschlagwortungsobjekte vor der Anzeige bearbeiten. Als Parameter erhält man die Parent-Objekt-Id und die Liste der Sord Objekte. Die Parent-Id ist größer als 0, wenn es sich um eine Child-Liste im Tree handelt. Wurde die Liste als Trefferergebnis einer Suche erzeugt, ist die Parent-Id < 0.

Das folgende Skript markiert alle Einträge mit der Farbkennung „Rot“ die den Text „ELO“ enthalten. Diese Kennung wird nur temporär und nur lokal aufgebracht, sie wird nicht im Archiv abgespeichert.

```
function eloCollectListStart(objid, sords, items) {
  for (var i = 0; i < sords.length; i++) {
    if (sords[i].name.contains("ELO")) {
      sords[i].kind = 1;
    }
  }
}
```

Wenn man nur die Such-Treffer markieren möchte, nicht aber die Child-Listen in der Baumanzeige, kann man den Parameter objid auswerten:

```
function eloCollectListStart(objid, sords, items) {
  if (objid < 0) {
    for (var i = 0; i < sords.length; i++) {
      if (sords[i].name.contains("ELO")) {
        sords[i].kind = 1;
      }
    }
  }
}
```

Bis zur Version 9 wird dieses Event nur für die Listen im Baum und in der Verschlagwortungssuche aufgerufen. Ab der Version 10 wird es auch für die Volltextsuche aktiviert. Da es hier weitere Informationen gibt, wird nun ein dritter Parameter „items“ angeboten. Dieser enthält analog zum sords Array eine Liste mit FindByFulltextResultItem Einträgen. Diese enthalten ebenfalls den sord Eintrag mit der Verschlagwortung, aber auch einen relevance Eintrag oder den summaryFulltext mit der Trefferumgebung.

Bei einer Verschlagwortungssuche ist der items Parameter „undefined“, genauso wie beim Blättern im Tree. Er wir nur bei einer Volltextsuche gefüllt.

Das folgende Beispiel erhöht die relevance aller Treffer, die mit dem Text „AW“ beginnen, um 100 Punkte:

```
function eloCollectListStart(objid, sords, items) {
  if (items) {
    for (var i = 0; i < items.length; i++) {
      var item = items[i];
      if (item.sord.name.startsWith("AW")) {
        item.relevance += 100;
      }
    }
  }
}
```

[10] PRÜFEN, OB EINE DATEI IM ARCHIV VORHANDEN IST

Ab der Version ELO 10 kann man Skriptfunktionen über die Kommandozeile aufrufen. Hierfür bietet das Programm EloActivateJC.exe einen speziellen Parameter an: rsf.

Wenn das Programm so aufgerufen wird: EloActivateJC.exe rsf <Skriptname> <Parameter>, dann führt es über die COM Schnittstelle einen Aufruf der Funktion runScriptFunction durch. Es wird die Skriptfunktion „<Skriptname>“ mit dem String Parameter „<Parameter>“ aktiviert. Somit kann man Skripte auch aus Umgebungen auslösen, die keine COM Programmierung erlauben.

Als Beispiel wird gezeigt, wie man im Windows Explorer ein Kontextmenüeintrag hinterlegen kann, der zu einer Datei prüft, ob diese bereits im Archiv vorhanden ist. Kontextmenüeinträge können im Windows Explorer über einen Registry Eintrag erzeugt werden. Hierzu trägt man im RegEdit unter HKEY_CLASSES_ROOT*\shell einen Schlüssel mit dem Namen des Kontextmenüeintrags ein – im Beispiel „Check ELO“. Darunter legt man einen weiteren Schlüssel Command an und dieser bekommt als Standardwert den Programmaufruf, der bei der Aktivierung des Menüeintrags ausgeführt werden soll.

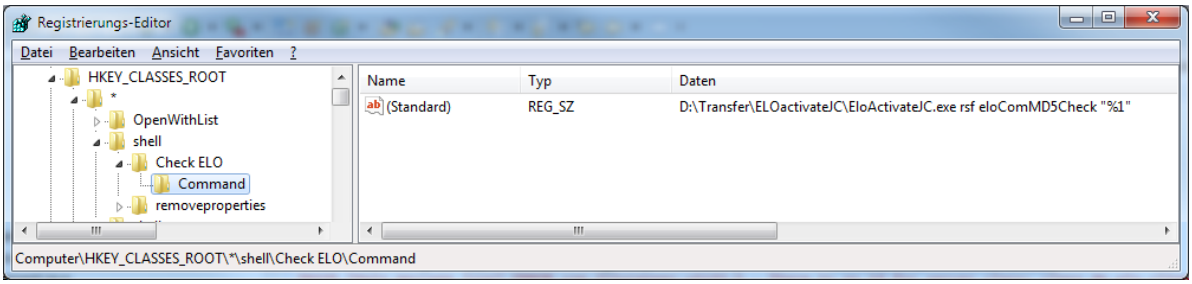


Abbildung 33: Menü-Eintrag in der Registry

```
Windows Registry Editor Version 5.00

[HKEY_CLASSES_ROOT\*\shell\Check ELO]

[HKEY_CLASSES_ROOT\*\shell\Check ELO\Command]
@="D:\\Transfer\\ELOactivateJC\\EloActivateJC.exe rsf eloComMD5Check \"%1\""
```

Wenn der Anwender nun das Kontextmenü im Explorer über einer Datei öffnet, findet er einen zusätzlichen Eintrag Check ELO vor. Wenn er diesen Eintrag anklickt, wird im Java Client das Skript eloComMD5Check aktiviert. Als Parameter wird der Dateiname mitgegeben.

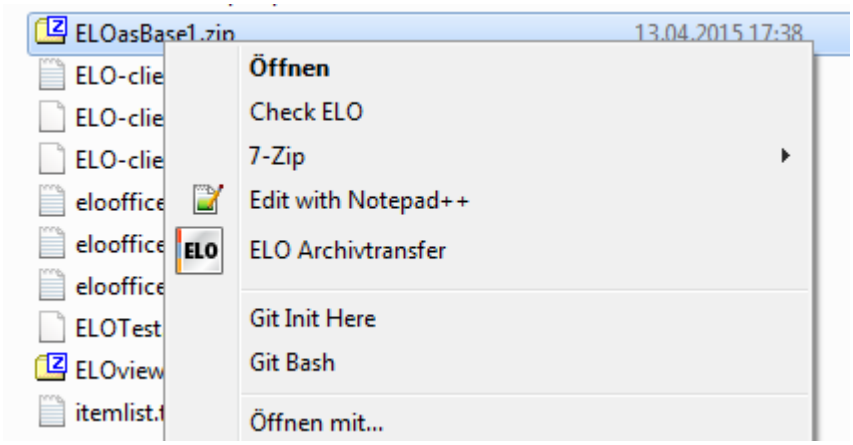


Abbildung 34: Kontextmenüeintrag Check ELO

Als Nächstes muss im Java Client noch das Skript eloComMD5Check angelegt werden. Dieses Skript liest den Dateinamen aus dem Parameter ein und führt eine MD5 Suche nach dem Dateiinhalt im Archiv durch. Praktischerweise gibt es dafür eine fertige Utils-Funktion: utils.findByMD5. Wenn die Datei bereits im Archiv vorhanden ist, führt das Skript gleich noch ein gotoId zu diesem Dokument aus.

```
function eloComMD5Check(param) {
  var sord = utils.findByMd5(new File(param));
```

```
  if (sord) {
    workspace.gotoId(sord.id);
  } else {
    workspace.setFeedbackMessage("Nicht im Archiv vorhanden");
  }
}
```

Beachten Sie bitte, dass aus Sicherheitsgründen nur Skriptfunktionen über die COM Schnittstelle aufgerufen werden können, die mit dem Präfix eloCom beginnen. Damit soll sichergestellt werden, dass externe Programme nicht beliebige interne Skriptfunktionen des Java Clients aktivieren können.

[10] ANWENDERORDNER „PERSÖNLICHER BEREICH“ FINDEN

Der Anwenderordner eines ELO Anwenders wird beim Programmstart automatisch in die Schrankliste referenziert. Wenn man darauf per Skript zugreifen will, kann man die ELO Objekt-Id des Ordners über die Skriptfunktion workspace.userNodeId abrufen. Der Ordner „Persönlicher Bereich“ ist ein Unterordner, den man wiederum über archive.getElementByArcpathRelative erreicht.

```
  var id = workspace.userNodeId;
  var dataItem = archive.getElementByArcpathRelative(id, "¶Persönlicher Bereich ");
```

[10] DUBLETTENPRÜFUNG IM ARCHIV

Wenn man per Skript eine Datei ablegen möchte, ist es manchmal notwendig zu prüfen, ob das Dokument bereits im Archiv vorhanden ist. Hierzu gibt es im UtilsAdapter einen Aufruf, der das extrem einfach macht: utils.findByMd5(file).

Dieser Aufruf ermittelt die MD5 Checksumme der Datei und führt im Archiv eine Suche nach dieser Checksumme durch. Wenn das Dokument noch nicht vorhanden ist, liefert der Aufruf null zurück. Ist es bereits vorhanden, liefert der Aufruf das Sord Objekt zum ersten Treffer zurück.

Ein null Return Wert sagt genau genommen nicht, dass die Datei noch nicht im Archiv ist. Er sagt nur aus, dass es keine Version dieser Datei im Archiv gibt, die für den Anwender sichtbar ist.

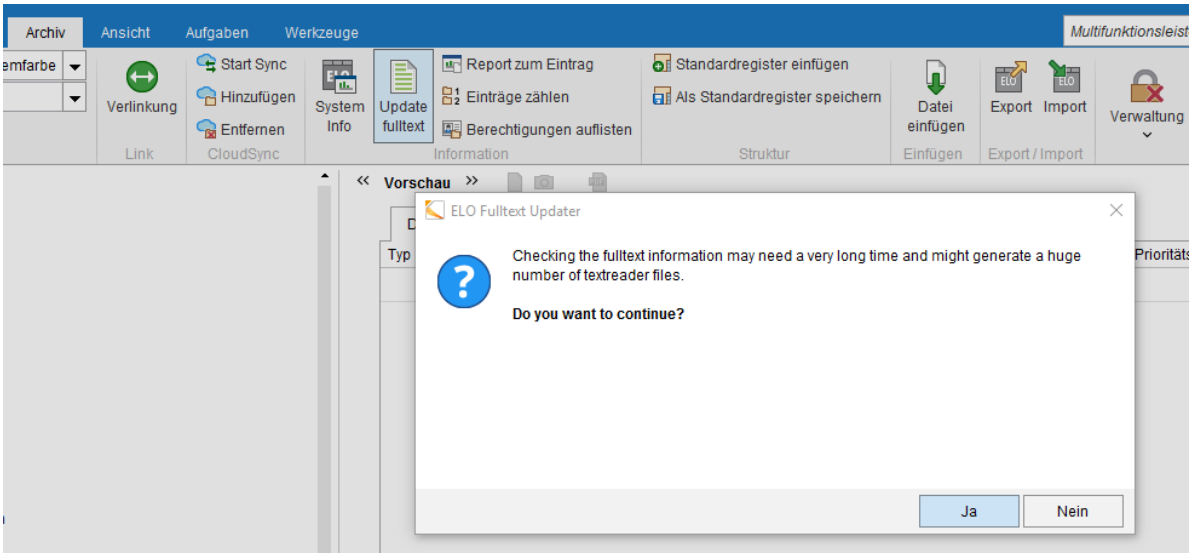
[10.2] VOLLTEXT-INFORMATION ERNEUERN

Manchmal kommt es vor, dass man Dokumente in den Volltext aufnimmt, daraus aber keine Textinformation extrahiert werden kann. Das kann an einem Konfigurationsfehler liegen oder daran, dass der Textreader dieses Dokumentenformat nicht verarbeiten kann. Wenn man das Problem behoben hat, möchte man gerne die betroffenen Dokumente erneut in die Volltextverarbeitung übergeben, in der Hoffnung, dass diese nun korrekt verarbeitet werden und die Volltextinformation gespeichert werden kann.

Das folgende Skript läuft über alle Untereinträge des aktuell selektierten Ordners oder Dokument und prüft, ob diese zur Volltextverarbeitung angemeldet sind. Wenn ja, wird im nächsten Schritt die Größe der Volltextdatei geprüft. Wenn diese einen im Skript voreingestellten Wert von 30 unterschreitet (z.B. weil nur „notConv“ eingetragen wurde), wird dieses Dokument aus dem Volltext entfernt und neu eingetragen. Das führt dazu, dass die Dokumentendatei erneut an den Textreader zur Verarbeitung übergeben wird.

Beachten Sie bitte, dass dieses Skript bei sehr umfangreichen Ordnern eine sehr lange Laufzeit benötigt. Zudem können Sie sich eine große Anzahl von Dateien im Textreader-Bereich erzeugen, die alle gleichzeitig zur Verarbeitung übergeben werden. Damit können Sie insgesamt eine sehr hohe Serverlast erzeugen, die den normalen Betrieb stören könnte.

Bei umfangreichen Archivstrukturen sollten Sie die Kontrolle deshalb in mehrere Teilschritte aufteilen. Zudem sollten Sie dieses Skript über die Berechtigungen so einschränken, dass nur Administratoren oder eingewiesene Mitarbeiter dieses ausführen können.




```
function getScriptButton844Name() {
  return "Update fulltext";
}

function getScriptButtonPositions() {
  return "844,archive,information";
}
```

```
function eloScriptButton844Start(){
  startUpdate();
}

var refreshCount;
var documentCount;

function startUpdate() {
  refreshCount = 0;
  documentCount = 0;
  var root = workspace.activeView.firstSelected;
  var msg = "<html><p>Checking the fulltext information may need a very long time and might
generate a huge number of textreader files.</p> <p><b>Do you want to
continue?</b></p></html>";
  if (workspace.showQuestionBox("ELO Fulltext Updater", msg)) {
    walk(10, root);
    msg = "<html><p><b>Update fulltext completed</b></p> <p>Documents refreshed:
" + refreshCount + "</p><p>Documents processed: " + documentCount + "</p></html>";
    workspace.showInfoBox("ELO Fulltext Updater", msg);
  }
}

function walk(level, root) {
  if (level < 0) {
    return;
  }

  if (root.isDocument()) {
    processDocument(root);
  } else {
    var items = root.getChildrenList();
    for (var i = 0; i < items.size(); i++) {
      var item = items.get(i);
      walk(level - 1, item);
    }
```

```
  }
}

function processDocument(root) {
  documentCount++;
  if (root.sord.details.fulltext) {
    var editInfo = ixc.checkoutDoc(root.id, null, EditInfoC.mbSordDoc, LockC.NO);
    if ((editInfo.document.docs.length > 0) && editInfo.document.docs[0]) {
      var size = editInfo.document.docs[0].fulltextContent.size;
      if (size < 30) {
        log.info("refresh fulltext (" + size + ") of: " + root.name);
        refreshCount++;
        root.sord.details.fulltext = false;
        root.saveSord();
        root.sord.details.fulltext = true;
        root.saveSord();
      }
    }
  }
}
```

Toolbar und Navigation im Java Client

Der Navigationsbereich im Java Client kann neben den Standard-Arbeitsbereichen wie z.B. Archiv oder Postbox auch um aufgabenspezifische Bereiche erweitert werden.

INTRANET SEITE IM JAVA CLIENT EINBLENDEN

Der Java Client bietet die Möglichkeit, den Navigationsbereich um zusätzliche aufgabenbezogene Einträge zu erweitern. In vielen Firmen gibt es ein Intranet mit wichtigen Informationen und Aktivitäten. Diese Seite kann man im Java Client als Navigationsbereich eintragen. Hierzu ist aber der Version 9.3 noch nicht mal ein Skript notwendig, es funktioniert über einen Konfigurationseintrag.

Im Administration Ordner gibt es einen Unterordner „ELOapps“ und dort wiederum „ClientInfos“. Hier können Sie die Konfiguration für die Intranet Seite in Form eines JSON Objekts hinterlegen.

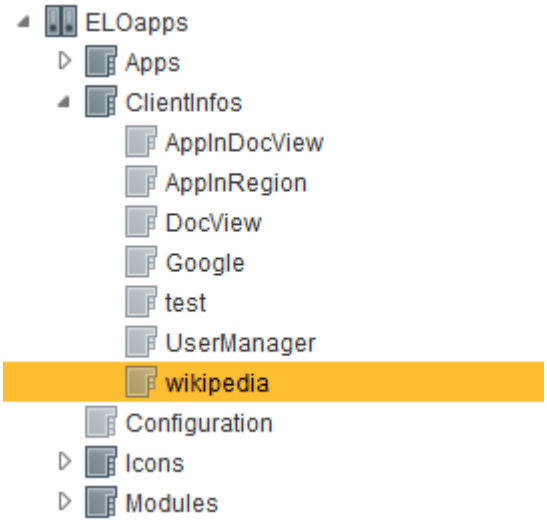


Abbildung 35: Pfad zum Konfigurationsbereich für die Navigation

Dazu legen Sie unter ClientInfos einen Ordner an und tragen den JSON Text in das Memo Feld ein:

```
{
  "selector": {
    "ids": []
  },
  "view": {
    "type": "REGION"
  },

  "web": {
```

```
    "eloSession": false,
    "url": "https://de.wikipedia.org/wiki/Wikipedia:Hauptseite",
    "name": "Wikipedia"
  },

  "id": "wikipedia"
}
```

Diese Definition legt fest, dass ein neuer Navigationsbereich angelegt werden soll (view – type = REGION) mit dem Namen „wikipedia“ (id = wikipedia) und in diesem Bereich soll ein Browserfenster (web) eingeblendet werden mit der Wikipedia Url (url = https://de.wikipedia.org...).

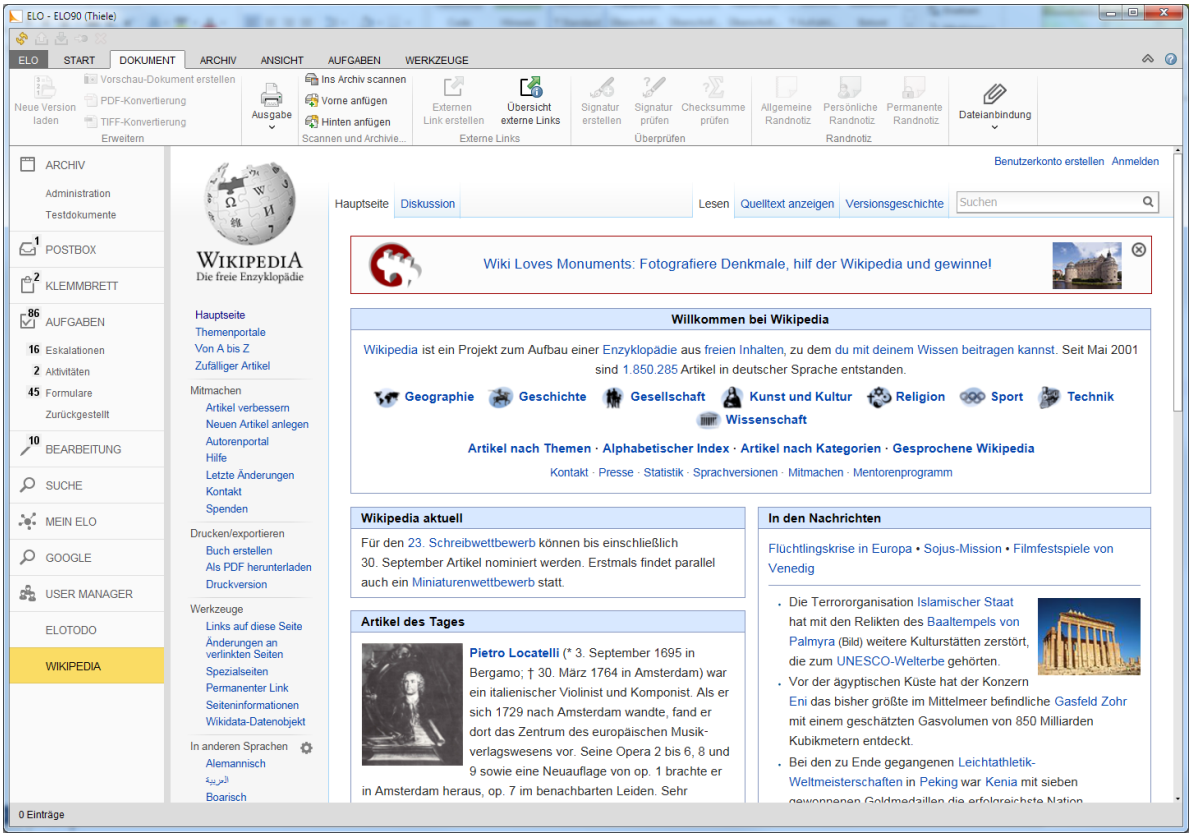


Abbildung 36: Client mit Navigationsbereich WIKIPEDIA

Bei älteren Clientversionen kann man dieses durch ein Skript erreichen, welches die notwendigen Schritte (Bereich einblenden, Browserfenster erzeugen, URL laden) durchführt. Ab der Version 9.3 sollte man dieses Skript aber nicht mehr nutzen und stattdessen besser die oben beschriebene Konfiguration verwenden.

```
const baseUrl = " https://de.wikipedia.org/wiki/Wikipedia:Hauptseite";
const viewName = "ELO Wiki";
const viewIconGuid = "(1CF19062-B91F-7D9E-4DE6-2C46883FE45C)";

function eloWorkspaceStarted() {
  extraViewBrowser.showBrowser(baseUrl);
}

function eloScriptsReloaded() {
  extraViewBrowser.showBrowser(baseUrl);
}

function eloRefreshViewStart() {
  extraViewBrowser.refreshUrl();
}

var extraViewBrowser = {

  browserToolbar : null,

  refreshUrl: function() {
    var view = workspace.activeView;
    if (view.name == viewName) {
      this.browserToolbar.navigate("");
      this.browserToolbar.navigate(baseUrl);
    }
  },

  showBrowser: function(url) {

    if (!this.browserToolbar) {
      this.browserToolbar = components.createBrowser();
      var controlPanel = components.createToolbar(0);
      workspace.removeView(viewName);
      var view = workspace.addView(viewName, false, this.browserToolbar, controlPanel);
```

```
      view.setHasPreview(false);

      try {
        var icon = archive.getElementByGuid(viewIconGuid);
        view.setIcon(icon);
      } catch(e) {
        log.info("Cannot find view icon: " + e);
      }
    }
    this.browserToolbar.navigate(url);
  }
}
```

Dokumentenanzeige über den Windows Protocol Handler

Unter Windows kann sich eine Applikation als Protocol Handler anmelden und bestimmte Aktionen ausführen, wenn ein Link mit diesem Protokoll aktiviert wird. Das wird z.B. für „mailto:“ URLs verwendet.

Die ELO Anwendung ELOactivate meldet sich selber als Protocol Handler für das Präfix elodms an. Wenn man also eine URL der Form elodms://... erzeugt, wird der Browser ELOactivate aufrufen, wenn ein Anwender so einen Link aktiviert. Damit kann man aus Web-Seiten oder Emails heraus den ELO Client aufrufen und die Dokumentenanzeige aktivieren.

Folgende Optionen stehen zur Verfügung:

elodms://([guid])

elodms://wf/[flowId]/[nodeId]

Die [guid] ist eine ELO Objekt GUID und in diesem Fall springt der Client auf diesen Eintrag

Die [flowId] und [nodeId] ist eine Workflow-Id und Knoten-Id. Wenn der Anwender diesen Eintrag in seiner Aufgabenliste hat, springt der Client auf diese Aufgabe.

Ab der Client Version 9.2 gibt es für die GUID Variante noch zwei Optionen:

elodms://([guid])@P[Seitennummer]

elodms://([guid])@A[Annotation-Id]

Damit können Sie direkt in ein Dokument reinspringen und eine vorgegebene Seite aufblättern. Sie zweite Variante mit einer Annotation-Id dient z.B. zum Anspringen von Lesezeichen.

BUTTON MIT TOOLTIP ERSTELLEN

Die Erstellung eines Buttons im Java Client ist recht einfach. Es müssen im Prinzip drei Funktionen implementiert werden: getScriptButtonXYZName(), eloScriptButtonXYZStart() und getScriptButtonPositions().

```
function getScriptButton100Name() {
  return "Projektantrag";
}

function getScriptButtonPositions() {
  return "100,home,navigation";
}

function eloScriptButton100Start(){
  // .. my Code
  workspace.setFeedbackMessage("Button pressed”);
}
```

Wenn man jetzt noch einen Tooltip hinterlegen möchte, kann man das über eine weitere Funktion getScriptButtonXYZTooltip() erledigen. Diese Funktion gibt als Return-Wert den gewünschten Text zurück. Sie können den Text über in mehrere Absätze unterteilen. Leider sind keine weiteren HTML Formatierungen möglich.

```
function getScriptButton100Tooltip(){
    return "Ein Abschnitt mit Erläuterungen</p><p>Noch ein Abschnitt";
}
```

[10] VERSCHLAGWORTUNGSABHÄNGIGE ANZEIGE

Über den AppManager kann man die bevorzugte Anzeige in Abhängigkeit von der Verschlagwortungsmaske einstellen. Das funktioniert natürlich nur, wenn der gewünschte Tab in der aktuellen Ansicht vorhanden ist. Für ein Dokumententyp, der alle wichtigen Informationen in einem Formular anzeigt, kann man die Ansicht auf das Formular umstellen – alle anderen Maskentypen bleiben auf der Dokumentenansicht.

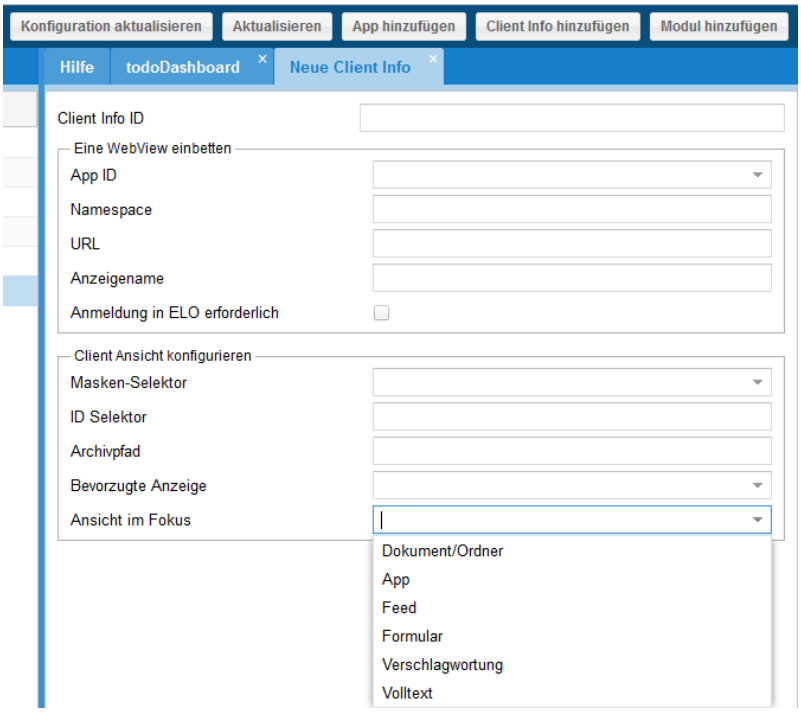


Abbildung 37: AppManager - Umstellung der Ansicht

Wenn man die Ansicht in Abhängigkeit von der aktuellen Verschlagwortung durchführen will, z.B. in Abhängigkeit des Status oder eines Wertes in der Kurzbezeichnung, benötigt man ein Skript. Dieses Skript prüft die Bedingung und wählt daraufhin den passenden Anzeige-Tab aus.

```
importPackage(Packages.de.elo.client.scripting.constants);

function eloPreviewAvailable(mode, actElement){
  var maskAssignments = {};
  maskAssignments["To do item"] = CONSTANTS.PREVIEW_CONTENT.FORMULAR;
  maskAssignments["EMail"] = CONSTANTS.PREVIEW_CONTENT.FEED;
  maskAssignments["Ordner"] = CONSTANTS.PREVIEW_CONTENT.INDEX_PREVIEW;

  try{
    if(String(mode) == "SINGLE_SELECTION"){
      var actMaskName = String(actElement.docMaskName);
      var viewMode = maskAssignments[actMaskName];

      if (actElement.name.contains("2")) {
        viewMode = CONSTANTS.PREVIEW_CONTENT.FULLTEXT;
```

```
    }

    if (!viewMode) {
      viewMode = CONSTANTS.PREVIEW_CONTENT.DOCUMENT_PREVIEW;
    }

    preview.switchToTab(viewMode);
  }
 }catch(e){
  log.info("error changing viewMode");
 }
}
```

[10] NAMEN DER ANSICHTEN

Jede Ansicht hat einen Standardnamen – Search, Task, Clipboard, Archive…

Bei den Such- und Task-Ansichten hat man aber das Problem. dass es davon mehrere Ansichten geben kann. Deshalb gibt es neben dem Propery view.name auch ein view.viewName, welches den speziellen Namen zurückliefert. Dieser wurde beim Erzeugen des Views vom Skript oder Anwender vergeben.

```
var views = tasksViews.tasksViews;
for (var i = 0; i < views.size(); i++) {
  var view = views.get(i);
  log.debug(view.viewName);
}
```

[10] SUCHBEGRIFF AUTOMATISCH IN DIE TREFFERANZEIGE ÜBERNEHMEN

Seit der Version 9.3 besitzt der Java Client ein eigenes Suchfeld in der Tiff- und PDF Anzeige. Wenn der Anwender hier einen Text eingibt, werden die jeweiligen Trefferstellen im Dokument markiert. Da es nun ein eigenes Feld gibt, wird nicht automatisch der Suchbegriff zur Anzeige verwendet. Der Anwender kann durch ein Doppelklick in das Textfeld den Begriff manuell übernehmen – oder man trägt ihn per Skript automatisch ein.

```
function eloSearchResultAvailable(view, findResult) {
  try {
    var term = view.findInfo.findDirect.query;
    preview.highlightText = term;
    log.info(term);
```

```
    } catch(e) {
    }
  }
```

Gegenüber älteren Versionen hat die aktuelle Client Version den Vorteil, dass der Skriptaufruf als Parameter die aktuelle Ansicht und die Ergebnisliste enthält. Man kann im Skript also sicher zuordnen, zu welchem View der Aufruf gehört und wie die Suchparameter waren und welche Ergebnisse zurückgeliefert wurden.

[10.1] Funktionsaufruf per Hotkey (Click&Find)

Ab der Version 10.1 können Sie die Ausführung von Skriptfunktionen an Windows Hotkeys binden. Hierzu fügen Sie in die Skriptdatei in das Event eloWorkspaceStarted die Funktionaufrufe workspace.addHotKey ein. Der erste Parameter gibt den Namen der Taste an, der zweite Parameter den Namen der aufzurufenden Funktion.

```
unction eloWorkspaceStarted() {
  workspace.addHotKey("alt T", "hello1");
  workspace.addHotKey("alt Z", "hello2");
}

function hello1() {
  var result = utils.transferByClipboard(true, true);
     workspace.showInfoBox("ELO", "HELLO T: " + result);
}

function hello2() {
  var result = utils.transferByClipboard(false, true);
     workspace.showInfoBox("ELO", "HELLO Z: " + result);
}
```

Wenn der Anwender nun die Taste <ALT>-T drückt, wird die Funktion hello1 aufgerufen. Dort kommt eine weitere praktische Hilfsfunktion zum Einsatz: utils.transferByClipboard. Diese Funktion simuliert einen Maus-Doppelklick, wenn der erste Parameter true ist. Anschließend wird ein <STRG>-C (copy) Aufruf simuliert, wenn der zweite Parameter auf true steht. Die Kombination Doppelklick selektiert in den meisten Anwendungen das Wort unter dem Mauscursor. <STRG>-C kopiert den selektierten Wert in das Windows Clipboard. Die Funktion liest am Ende das Clipboard aus und gibt den markierten Wert als Rückgabewert an das aufrufende Skript zurück. Dieser Wert kann dann z.B. für eine Suche verwendet werden.

Diese Funktionen kann man dazu nutzen, die Click&Find Funktionen direkt im Java Client, ohne weitere Hilfsprogramme, auszuführen. Wenn man per Hotkey z.B. die Versionsnummer aus dem ReleaseRequest Formular suchen möchte, könnte das Skript dazu so aussehen.

```
function hello1() {
  var result = utils.transferByClipboard(true, true);
  workspace.setFeedbackMessage("Search Version: " + result);
  var view = searchViews.getSearchView("RelReq");
  if (!view) {
    view = searchViews.addSearchView("RelReq");
  }

  var key = new ObjKey();
  key.name = "RR_VERSION";
  key.data = [result + "*"];

  var fbi = view.createFindByIndex();
  fbi.maskId = 52;
  fbi.objKeys = [key];

  var fi = view.createFindInfoByDate("");
  fi.findByIndex = fbi;
  view.search(fi, 1000, true);
}
```

[11.0] WORKFLOW-AUFGABE ANZEIGEN

Das workspace Objekt besitzt nun eine Methode gotoWorkflow(flowId, nodeId), welche es erlaubt, per Skriptbefehl auf einen bestimmten Workflow in der Aufgabenansicht zu springen. Der Parameter für die nodeId ist optional – er kann auf 0 gesetzt werden. In diesem Fall wird der erste Workflowknoten aus der Aufgabenansicht aktiviert, der zum angegebenen Workflow gehört.

Verschlagwortung im Java Client

In diesem Bereich werden einige Beispiele aufgeführt, die bei der Verschlagwortung von Dokumenten im Java Client helfen.

EINGABEFELD BESCHREIBBAR MACHEN

Der Maskeneditor bietet die Möglichkeit, Eingabefelder unsichtbar zu machen (z.B. für technische Werte, die für den Anwender nicht interessant sind) oder mit einem Schreibschutz zu versehen, damit keine irrtümlichen Änderungen vorgenommen werden können.

Wenn es jetzt im Prozess notwendig wird, dass ein schreibgeschütztes Feld doch mal geändert werden soll, muss man nicht gleich komplett auf den Schreibschutz verzichten. Man kann ihn auch gezielt per Script zurücknehmen.

```
function eloIndexDialogSetDocMask() {

   var maskName    =   "SupportEvent";
   var objKeyName  =   "ReadOnlyFeld";

   if( indexDialog.getDocMaskName().equals(maskName)) {
      //  Entsperren des Feldes
      indexDialog.getObjKey( objKeyName ).setEnabled(true);
   }
}
```

Bitte beachten Sie, dass diese beiden Feldeigenschaften „Unsichtbar“ und „Schreibgeschützt“ reine Komfortmaßnahmen und keine Sicherheitsmaßnahmen sind, da der interessierte Anwender beide Einstellungen mit etwas Aufwand umgehen kann.

AUTOMATISCHES ZÄHLERFELD

Der Indexserver bietet automatische Counter an, die einen eindeutigen hochgezählten Wert bieten. Auch dann, wenn mehrere Clients gleichzeitig einen Wert anfordern. Diese Zähler kann man z.B. für eine Rechnungsnummernvergabe verwenden. Im Java Client kann man einen Zählerwert über die Methode workspace.incCounter anfordern. Als Parameter werden der Counter-Name sowie der Initial-Wert für noch nicht vorhandene Counter angegeben.

Wenn man sicherstellen möchte, dass man fortlaufende Nummern hat, muss man dafür sorgen, dass diese nicht verworfen werden können. Wenn man z.B. das Rechnungsnummernfeld beim Betreten der Verschlagwortung füllt und der Anwender die Maske mit Abbruch verlässt, hat man eine ungenutzte Nummer. Bei Rechnungen ist das ziemlich schlecht, das gibt Probleme mit dem Finanzamt.

Sicherer ist es, wenn die Nummer erst beim Speichern mit „Ok“ eingetragen wird. Hier kann der Anwender nicht mehr abbrechen.

```
function eloIndexDialogOkStart() {
  if (indexDialog.docMaskId == 2) {
    var counterValue = indexDialog.getObjKeyValue("ELOOUTL1");
    if (counterValue == "") {
      var cnt = workspace.incCounter("cntELOOUTL1", 100);
      indexDialog.setObjKeyValue("ELOOUTL1", cnt);
      workspace.setFeedbackMessage("Zählerstand: " + cnt);
    }
  }
}
```

BUTTON HINTER EINER INDEXZEILE ANZEIGEN

Damit hinter einer Indexzeile ein Button eingeblendet werden kann, muss diese etwas verkürzt werden damit sie den Button nicht überdeckt. Anschließend muss der neue Button erzeugt werden und am Ende der Indexzeile angeordnet werden.

Für diese Funktion gibt es einen einfachen Aufruf addButton, der sowohl die Indexzeile kürzt, wie auch den Button einfügt. Der erste Parameter gibt die Button-Beschriftung an, der zweite Parameter den Namen der Callback-Funktion, wenn der Button vom Anwender geklickt wird und der dritte Parameter gibt die Breite des Buttons in logischen Einheiten an.

```
function eloIndexDialogSetDocMask() {
  if (indexDialog.docMaskName == "Lieferschein") {
    var liefer = indexDialog.getObjKey( 4 );
    liefer.addButton( "DoIt", "doButtonAction", 4 );
  }
}

function doButtonAction(){
  workspace.showInfoBox( "Test", "Button pressed" );
}
```

TAB AUSBLENDEN (1)

Der Java Client zeigt im Verschlagwortungsdialog einen oder mehrere Tabs für die Indexzeilen und weitere Tabs für den Zusatztext, die Optionen, Berechtigungen, Änderungen und weitere Infos an. Nicht immer sollen die Anwender alle Tabs sehen können. Dafür gibt es die Funktion setTabVisible in der Java Client API des Verschlagwortungsdialogs.

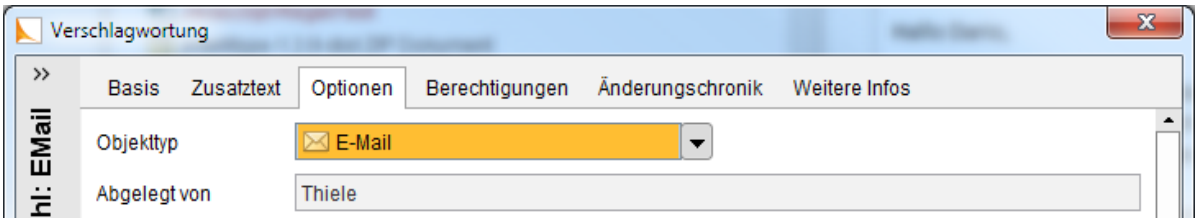


Abbildung 38: Normale Ansicht des Verschlagwortungsdialogs

Dieses einfache Beispiel blendet für einen bestimmten Anwender den Optionen Tab aus, wenn die Email Maske angezeigt wird.

```
const SPECIAL_MASK = 2;
const SPECIAL_USER = 3;
const OPTION_TAB = 11;

function eloIndexDialogSetDocMask() {
  if (indexDialog.docMaskId == SPECIAL_MASK) {
    if (workspace.userId == SPECIAL_USER) {
      indexDialog.setTabVisible(OPTION_TAB, false);
    }
  }
}
```

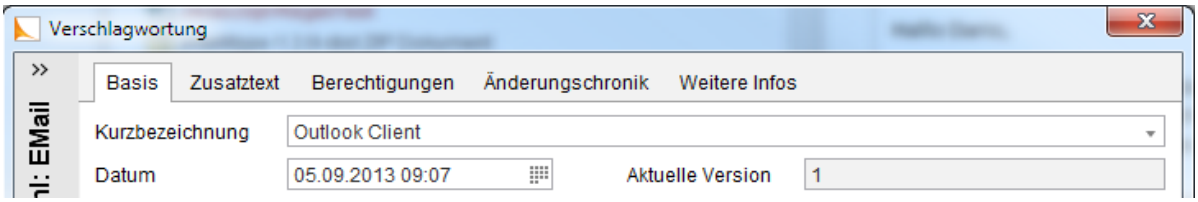


Abbildung 39: Verschlagwortung ohne Optionen Tab

TAB AUSBLENDEN (2)

Ein dynamisches Register enthält die Definition zur Datenabfrage im Zusatztext. Diese Information ist für die meisten Anwender im besten Falle irreführend. Im schlimmsten Falle versuchen Sie hier Änderungen vorzunehmen. Das kann man verhindern, indem man den Tab mit dem Zusatztext bei dynamischen Registern ausblendet.

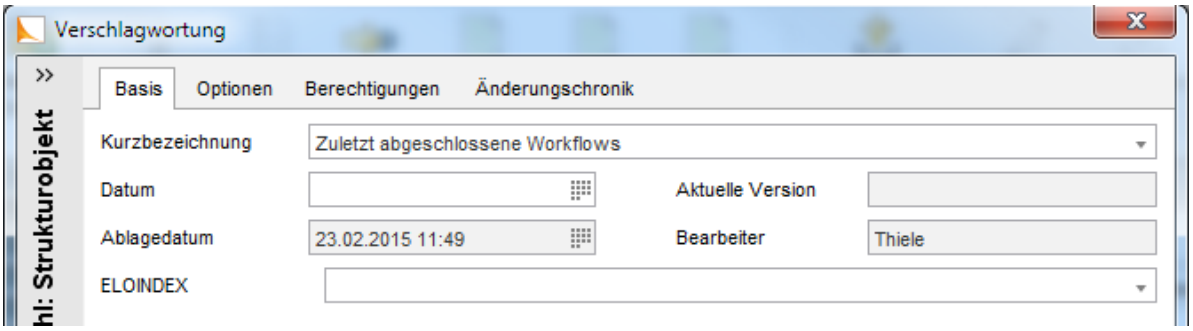


Abbildung 40: Verschlagwortung ohne Zusatztext

```
const MEMO_TAB = 10;

function eloIndexDialogSetDocMask() {
  if (!workspace.userRights.hasAdminRight()) {
    var desc = indexDialog.sord.desc;
    if (desc && (desc.length() > 0) && (desc.charAt(0) == 33)) {
      indexDialog.setTabVisible(MEMO_TAB, false);
    }
  }
}
```

Unter JavaScript gibt es unglücklicherweise zwei unterschiedliche String Typen: Einmal der normal JavaScript String, welcher bei allen skriptinternen Textoperationen entsteht. Und zum anderen der Java String, welchen Sie aus der ELO Java API des Java Clients, ELOas oder Indexserver bekommen. Es ist nicht immer leicht zu sehen, welchen Typ Sie erhalten und in den meisten Funktionen sind die beiden Typen auch identisch. Allerdings unterscheidet sich die Abfrage der Länge, welche in JavaScript mit myJavaScriptText.length und in Java mit myJavaString.length() erfolgt.

AUTOMATISCHES CHECKIN BEIM SPEICHERN

Streng genommen keine Skript-Aufgabe, da kein Skript benötigt wird. Der Java Client bietet die Möglichkeit, dass Sie in der Verschlagwortung eines Vorlagen-Dokuments eintragen können, dass dieses Dokument automatisch eingecheckt wird. Das funktioniert nur bei Dokumenten, bei denen das Bearbeitungsprogramm die Datei während der Bearbeitung geöffnet hält.

Der Java Client prüft in diesem Fall regelmäßig nach, ob die Datei im CheckOut Bereich exklusiv erreichbar ist. Wenn nein – dann ist noch die Originalanwendung geöffnet. Sobald die Datei exklusiv erreichbar ist, wird sie dann automatisch wieder eingecheckt.

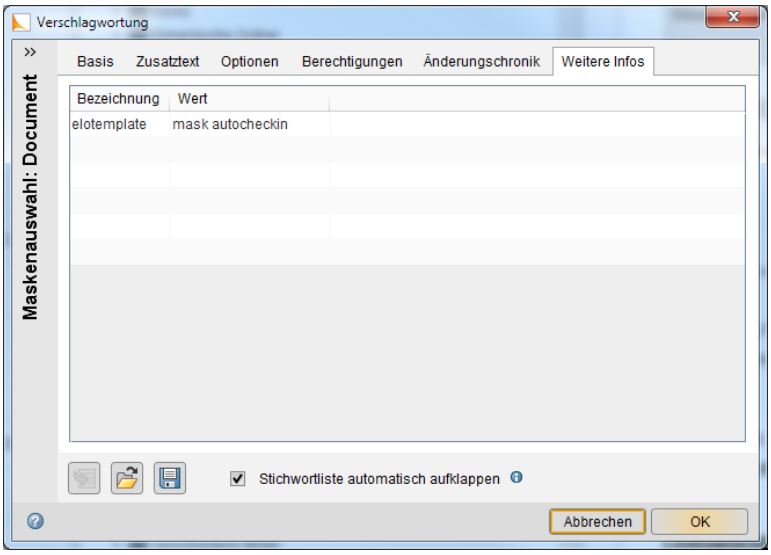


Abbildung 41: Marker für Autocheckin eintragen

Da man das nicht mit allen Dokumententypen machen kann oder will, muss jedes Auto-Checkin Dokument einzeln gekennzeichnet werden. Hierfür wird in der Verschlagwortung der Vorlage unter „Weitere Infos“ eine Zeile „elotemplate“ angelegt. Als Wert kann man dann „autocheckin“ eintragen – dann wird diese Option aktiv.

Weiterhin besteht noch die Möglichkeit „mask“ einzutragen. In diesem Fall wird für das neue Dokument nicht die Standardmaske für neue Dokumente, sondern die Maske und Verschlagwortung des Vorlagendokuments verwendet.

WIE WURDE DER VERSCHLAGWORTUNGSDIALOG GESCHLOSSEN

Wenn man per Skript die Anzeige der Verschlagwortung über indexDialog.editSord aufruft, bekommt man je nach Aufrufart die Information zurück, ob die Verschlagwortung geändert wurde (Aufruf mit drei Parametern) oder, ob der Dialog mit Ok geschlossen wurde (älterer Aufruf mit zwei Parametern).

Manchmal möchte man aber zwischen 3 Zuständen unterscheiden:

1. Per Ok geschlossen und verändert
2. Per Ok geschlossen aber keine Veränderung
3. Per Cancel geschlossen

Das kann man erreichen, indem die editSord Variante mit drei Parametern verwendet. Diese liefert zurück, ob der Eintrag verändert wurde. Damit kann man die Fälle 2 und 3 vom Fall 1 abgrenzen. Zusätzlich kann man anschließend die Methode indexDialog.wasClosedByOk aufrufen und damit den Fall 2 vom Fall 3 abgrenzen.

BILD IM VERSCHLAGWORTUNGSDIALOG ANZEIGEN

In einem Kundenprojekt kann die Frage auf, wie man im Verschlagwortungsdialog ein Bild einblenden kann, welches im gleichen Ordner liegt.

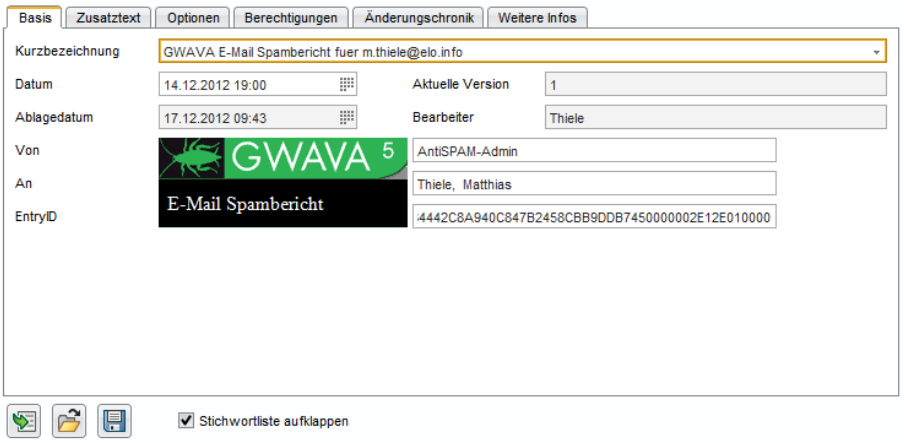


Abbildung 42: Verschlagwortung mit Bild

Im Java Client ist das relativ einfach – man hinterlegt ein Skript, welches bei der Anzeige des Verschlagwortungsdialogs aktiv wird. Wenn die richtige Maske angezeigt wird, sucht es im gleichen Ordner nach einem Eintrag mit der Kurzbezeichnung "Image" und lädt die Datei in ein Java ImageIcon Objekt. Dieses wird wiederum in ein JLabel eingepackt und das im Verschlagwortungsdialog an die freie Stelle geladen.

```
importPackage(Packages.javax.swing);

function eloIndexDialogSetDocMask() {
  var sord = indexDialog.sord;
  if (sord.maskName == "EMail") {
    var parentId = sord.parentId;
    if (parentId > 0) {
      try {
        var image = archive.getElementByArcpathRelative(parentId, "¶Image");
        if (image.isDocument()) {
          var file = image.file;
          var icon = new ImageIcon(file.path);

          var label = new JLabel();
          label.setIcon(icon);
          indexDialog.addComponent(1, 14, 4, 30, 3, label);
        }
      } catch(e) {
        log.debug("No Image: " + e);
```

```
    }
   }
  }
}
```

FREI DEFINIERBARE STICHWORTLISTEN

Im Maskeneditor können Sie zu einer Indexzeile eine passende Stichwortliste hinterlegen. Es kommt aber vor, dass diese nicht statisch festliegen, sondern situationsbezogen berechnet werden müssen. In diesem Fall können Sie zur Laufzeit im Java Client ein Array aus Strings zusammenstellen und dieses an die Indexzeile als Stichwortliste übergeben.

```
const FROM_FIELD = 0;
function eloIndexDialogSetDocMask() {
  if (indexDialog.docMaskId == 2) {
    var key = indexDialog.getObjKey(FROM_FIELD);
    var swl = [
              new Date(),
              workspace.userName,
              workspace.clientVersion
             ];
    key.setKeywords("Sondereinträge", swl);
  }
}
```

In diesem Beispiel besteht die dynamische Stichwortliste aus drei Einträgen: dem aktuellen Datum, dem aktuell angemeldeten Anwender und der Client-Versionsnummer. Alles Daten, die man nicht statisch über den Maskeneditor hinterlegen könnte.

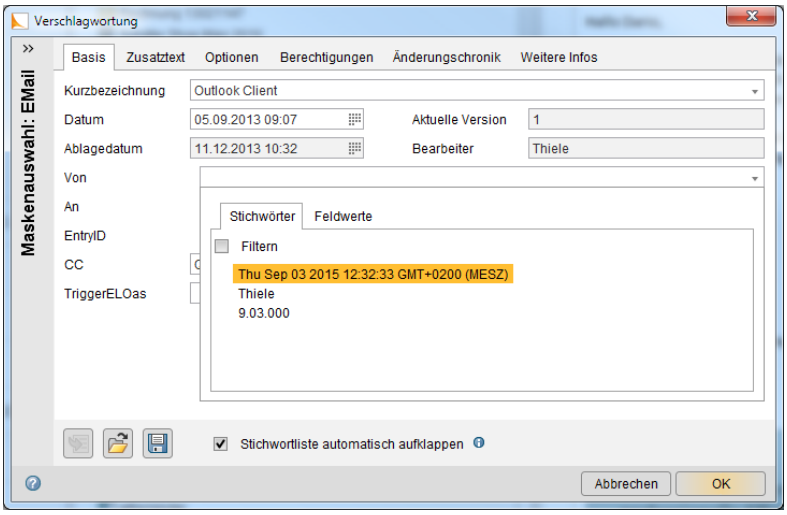


Abbildung 43: Anzeige der dynamischen Stichwortliste

STRING AN DAS WINDOWS CLIPBOARD ÜBERGEBEN

Zum Übergeben eines Textes aus dem Java Client an das Windows Clipboard kann man folgende Funktion verwenden (im Beispiel wird die Kurzbezeichnung des aktuell selektierten Eintrags verwendet):

```
importPackage(Packages.java.awt);
importPackage(Packages.java.awt.datatransfer);


function getScriptButton100Name() {
  return "Copy to Clipboard";
}

function getScriptButtonPositions() {
  return "100,home,view";
}

function eloScriptButton100Start(){
  var str = workspace.activeView.firstSelected.name

  var toolkit = Toolkit.getDefaultToolkit();
  var clipboard = toolkit.getSystemClipboard();
  var strSel = new StringSelection(str);
```

```
  clipboard.setContents(strSel, null);


}
```

ADRESSE IN GOOGLE MAPS ANZEIGEN

Dieses Skript liest die Adressinformation aus der aktuellen Verschlagwortungsmaske aus und erzeugt daraus eine URL zum Aufruf in Google Maps. Diese wird dann über Desktop.browse() an einen externen Browser zur Anzeige übergeben.

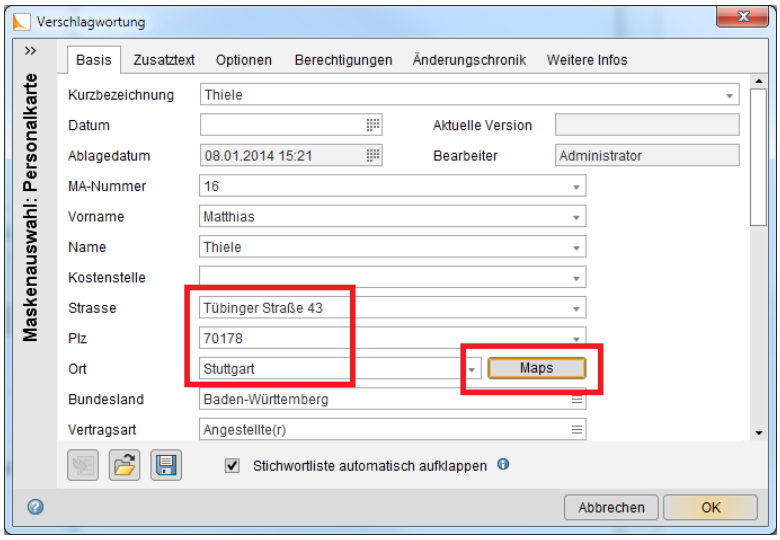


Abbildung 44: Verschlagwortung mit Adressfeldern

Es wird davon ausgegangen, dass in einer Ablagemaske die Postleitzahl, der Ort und die Straße hinterlegt sind. Diese Zeilen werden auf Button-Klick ausgelesen und zu einer Google Maps URL zusammengestellt. Diese wird dann an den Browser übergeben. Das Skript verwendet ein relativ neues Feature des Java Clients, welches erst ab der Version 9.1 verfügbar ist: Eine Indexzeile wird durch einen Skriptbefehl verkürzt und mit einem Skript-Button ergänzt (key.addButton).

Das Skript selber ist mehr oder weniger trivial. In der Funktion eloIndexDialogSetDocMask wird geprüft, ob die Maske "Personalkarte" (Id = 332 in meinem Testarchiv) aktiv ist. In diesem Fall wird die Indexzeile 6 für den Ort mit einem Button "Maps" ergänzt.

Wenn der Button geklickt wird, liest die Event-Funktion die Indexzeilen PLZ, Ort und Straße aus und erzeugt daraus eine URL. Man kann nicht direkt den String verwenden, da eine Adresse viele Zeichen enthält, die in einer URL nicht zulässig sind - deshalb der Umweg über den URLEncoder. Desktop.desktop.browse ruft die URL schließlich im Browser auf.

```
function eloIndexDialogSetDocMask() {
  if (indexDialog.docMaskId == 332) {
    var key = indexDialog.getObjKey(6);
```

```
    key.addButton( "Maps", "triggerViewMap", 10 );
  }
}

function triggerViewMap() {
  var address = indexDialog.getObjKeyValue( "PA_PLZ" ) + " " +
    indexDialog.getObjKeyValue("PA_ORT") + "," +
    indexDialog.getObjKeyValue("PA_STR");

  address = "https://www.google.de/maps/place/" + java.net.URLEncoder.encode(address,
"UTF-8");
  var mapURI = java.net.URI.create( address );
  Desktop.desktop.browse(mapURI);
}
```

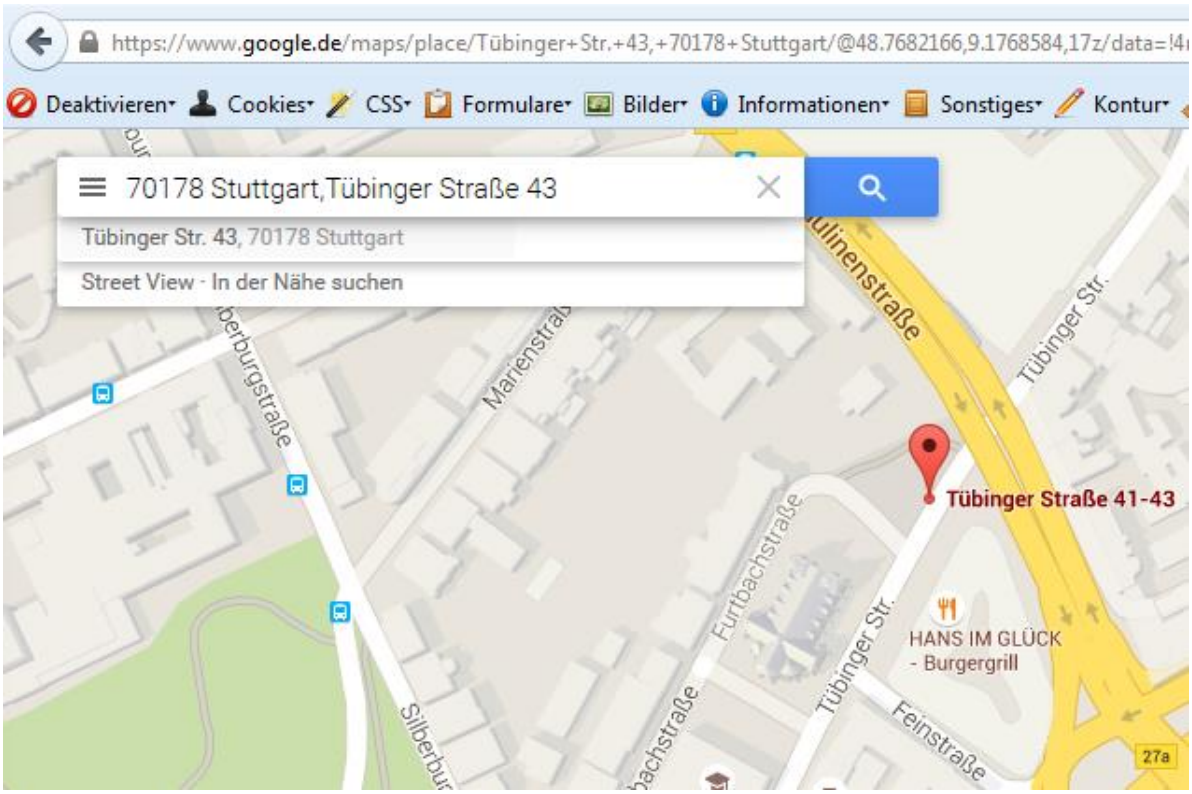


Abbildung 45: Anzeige in Google Maps

Man könnte das Skript noch universeller machen, indem man nicht gezielt auf eine Ablagemaske reagiert, sondern die Gruppennamen nach typischen Anzeichen einer Adresse durchsucht (PLZ, ORT,

STRASSE) und daraus die URL konstruiert. Allerdings wird man dann nicht immer die richtigen Felder erwischen.

[10] TREFFERLISTE BEARBEITEN

Man kann in der Trefferliste der Suche bestimme Einträge hervorheben indem man diese im Event „eloSearchResultAvailable“ prüft und bei Bedarf passend markiert. Die Markierung kann z.B. durch das Setzen der Farbe vorgenommen werden.

```
function eloSearchResultAvailable(view, findResult) {
  var sords = findResult.sords;

  for (var i = 0; i < sords.length; i++) {
    var item = sords[i];
    if (item.maskName == "ZRechnung") {
      item.kind = 3;
    }
  }
}
```

Beachten Sie bitte, dass es sich dabei nur um eine temporäre Änderung des aktuell angezeigten Eintrags handelt. Die Datenbankinformation wird dabei nicht verändert. Sobald der Anwender den Eintrag bearbeitet, wird wieder die ursprüngliche Version geladen.

Das Event ist nicht neu in der Version 10 – es ist auch schon in älteren Versionen enthalten. Dort allerdings ohne Parameter. Deshalb kann das Skript nicht feststellen, zu welcher Suchansicht es gehört und wird somit weniger nützlich.

[10] OCR CLICK-WORT IN SKRIPTDIALOGFELD ÜBERNEHMEN

Der Java Client bietet seit einiger Zeit die Möglichkeit, einen Begriff aus dem Dokument mit einem Klick in die Verschlagwortung zu übernehmen. Das kann man auch in eigene Dialoge übertragen. Hierzu muss an das Event „eloOcrWordClicked“ implementieren. Dieses Event wird dann getriggert, wenn der Anwender ein Wort im Dokument angeklickt hat und dieses per OCR erkannt wurde.

Im Skript Event muss nun noch das Ziel gefunden werden. Das ist nicht ganz einfach, weil zu dem Zeitpunkt des „Klicks“ der Fokus im Dokument und nicht in dem Textfeld liegt, welches das Wort bekommen soll. Java bietet allerdings in Dialogen ein Property „getMostRecentFocusOwner“ an. Dieses enthält das Dialog-Element, welches zuletzt den Fokus hatte.



Abbildung 46: Ausschnitt aus der Rechnung

Wenn der Anwender ein Wort im Dokument anklickt, wird das Event aktiviert. In dem Skript wird das aktuelle Input-Feld identifiziert und der Text dort eingetragen.

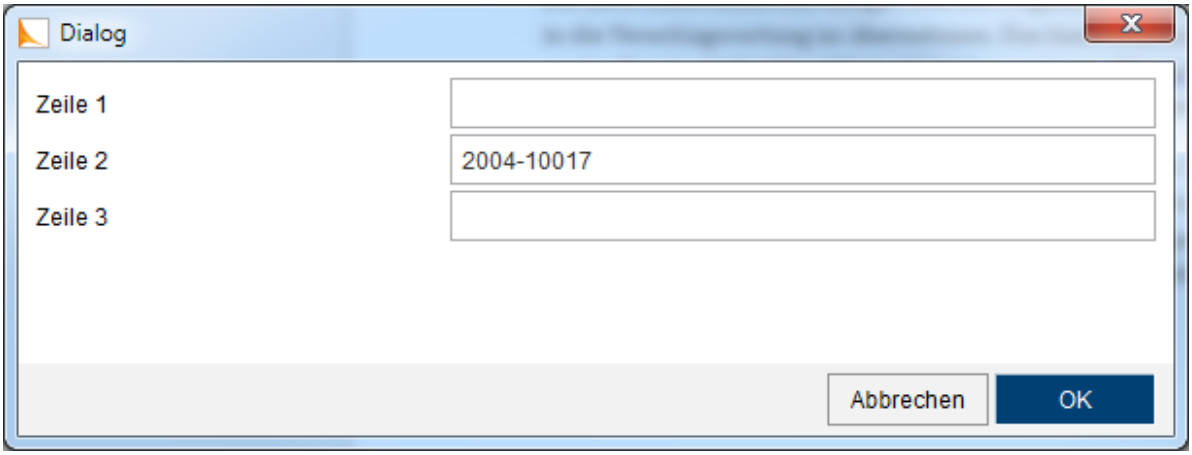


Abbildung 47: Feldinhalt nachdem die Belegnummer angeklickt wurde

```
var dialog = null;

function eloScriptButton100Start() {
  dialog = workspace.createGridDialog("Dialog", 3, 4);

  var gPanel = dialog.getGridPanel();
```

```
  gPanel.addLabel(1, 1, 1, "Zeile 1");
  var tf1 = gPanel.addTextField(2, 1, 2);

  gPanel.addLabel(1,2,1,"Zeile 2");
  var tf2 = gPanel.addTextField(2,2,2);

  gPanel.addLabel(1,3,1,"Zeile 3");
  var tf3 = gPanel.addTextField(2,3,2);

  var result = dialog.show("close", "close");
}

function close() {
  dialog = null;
}

function eloOcrWordClicked() {
  if (!dialog) {
    return;
  }

  try {
    var destination = dialog.getMostRecentFocusOwner();
    if (destination && (destination.class == "class javax.swing.JTextField")) {
      destination.setText(preview.clickedOcrWord);
    }
  } catch(e) {
    log.info("No destination or no setText method: " + e);
  }
}
```

[10] MAILS MIT ATTACHMENT MARKIEREN

Wenn man Mails über das ELO Makro ablegt, konnte man einstellen, dass Mails mit Attachment einen anderen Dokumententyp zugewiesen bekommen. Hierzu legt man in der Admin Console einen Dokumententyp „Mail mit Attachment“ an (mit einem passenden Icon).

Diese Anzeige kann man auch für Mails nutzen, die über die Postbox oder direkt per Drag & Drop ins Archiv abgelegt wurden. Hierfür benötigt man ein kleines Event-Script welches bei der Ablage neuer Dokumente den Dateityp prüft. Bei EML und MSG Dateien wird dann die Datei eingelesen und geprüft, ob sie Attachments besitzt. Wenn ja, wird dem Eintrag der neue Dokumententyp „Mail mit Attachment“ zugewiesen.

```
const MailWithAttachmentType = 260;

function eloInsertDocumentEnd(document) {
  var file = document.file;
  var ext = utils.getFileExtension(file).toLowerCase();
  if ((ext == "msg") || (ext == "eml")) {
    mail = utils.readMail(file);
    if ((mail != null) && (mail.attachments.length > 0)) {
      var sord = document.sord;
      sord.type = MailWithAttachmentType;
      document.sord = sord;
    }
  }
}
```

[10] NICHT-MODALER VERSCHLAGWORTUNGSDIALOG

Wenn man per Skript einen nicht-modalen Verschlagwortungsdialog aufrufen möchte, kann das Skript nicht darauf warten, dass der Anwender auf Ok oder Abbruch klickt. Denn dann wäre der Dialog Modal. Beim nicht-modalen Aufruf von indexDialog.editSord wird der Dialog angezeigt und der Skriptaufruf kehrt sofort zurück. An dieser Stelle endet das Skript dann auch typischerweise.

Damit das Skript auf Anwendereingaben reagieren kann, muss es beim Aufruf die Namen für die Event-Funktionen mitgeben, die beim Klick auf Ok und Abbruch ausgeführt werden sollen. Das sind zwei ganz normale JavaScript Funktionen die vom Java Client automatisch gestartet werden, sobald der Anwender den jeweiligen Button klickt.

```
 * Öffnet das SORD-Objekt zur Bearbeitung im Verschlagwortungs-Dialog. Wenn in
 * sord.access kein Schreibrecht gegeben ist, erscheint der Dialog im Read-Only-Modus.
```

```
  *
  * @param sord Zu bearbeitende Verschlagwortung
  * @param title Titel des Verschlagwortungsdialogs
  * @param okCallbackMethod Ein Event welches beim Klick auf "OK" aufgerufen werden
  *   soll oder null. Im Scripting kann das Schließen des Dialogs verhindert
  *   werden, wenn eine Funktion zu diesem Event einen negativen Rückgabewert liefert.
  * @param cancelCallbackMethod Ein Event welches bei "Abbrechen"
  *    oder "X" aufgerufen wird oder null. Das
  *    Schließen des Dialogs kann hierbei nicht abgebrochen werden.
  * @return True wenn das Sord geändert wurde, sonst False
  * @throws Exception
  * @since 10.00.000
```

```
function eloScriptButton100Start () {
  var item = workspace.activeView.firstSelected;

  indexDialog.editSord(item.sord, "Nicht-Modaler Test", "callbackOk", "callbackAbort");
  return -1;
}

function callbackOk() {
  workspace.setFeedbackMessage("Der Dialog wurde mit Ok beendet");
}

function callbackAbort() {
  workspace.setFeedbackMessage("Der Dialog wurde mit Abbrechen beendet");
}
```

[10.2] VERSCHLAGWORTUNG KOPIEREN

Im letzten Jahr wurde von einem Kunden moniert, dass man mit der F3 Funktion auch schreibgeschützte und unsichtbare Felder verändern kann. Das ist dem Anwender normalerweise nicht möglich. Gerade in prozessorientierten Umgebungen können hierdurch Fehler entstehen, da Prozessdaten aus einem Objekt unverändert an ein anderes Objekt übergeben werden. Aus diesem Grund mussten wir diese Funktion sperren, sobald unsichtbare oder schreibgeschützte Felder in der Ablagemaske waren.

Für Anwender, bei denen solche Fehler nicht auftreten können, war das ein unschöner Zustand: sie konnten ab dieser Version die Verschlagwortung nicht mehr einfach kopieren. Da wir nicht entscheiden können, zu welcher Gruppe der Anwender gehört, können wir direkt nichts an der Situation ändern. Ab der Version 10.2 gibt es aber eine Skriptschnittstelle, über die man im Projekt entscheiden kann, ob die Funktion zur Verfügung steht. Praktischerweise kann man die Entscheidung von der Verschlagwortung bzw. Ablagemaske abhängig machen. Bei allen unkritischen Masken wird das Kopieren erlaubt, ansonsten bleibt es verboten.

```
function eloEnableFillWithLastSord( actSord, lastSord, mode ) {
  if (indexDialog.docMaskName == "Bilderdemo") {
    indexDialog.fillEnabled = true;
    return -1;
  }
}
```

Damit man auch in prozessorientierten Umgebungen die Kopierfunktion anbieten kann, gibt es ein weiteres Event, welches es dem Skript erlaubt, zu entscheiden, was kopiert wird. In der Schleife über die ObjKeys kann man individuell entscheiden, ob der Inhalt kopiert wird, eventuell sogar verändert wird, oder ob die Zeile leer bleiben oder mit einem Default Wert gefüllt werden soll.

```
function eloFillWithLastSord( actSord, lastSord, findInfo ) {
  if ((lastSord != null) && (actSord.maskName == "Bilderdemo")) {
    indexDialog.name = "*** " + lastSord.name;
    var keys = lastSord.objKeys;
    for (var i = 0; i < keys.length; i++) {
      var key = keys[i];
      if (key.data[0]) {
        try {
          indexDialog.setObjKeyValue(key.name, "+ " + key.data[0]);
        } catch(e) {
          log.info("Copy error: " + e);
        }
      }
    }
    return -1;
  }
  return 0;
}
```

[10.2] VERSCHLAGWORTUNGSMASKE DURCH ANWENDERAUSWAHL BESTIMMEN

In einem Projekt hatten wir die Anforderung, dass bei der Ablage einer Datei per Drag&Drop der Verschlagwortungsdialog unterdrückt werden soll und stattdessen nur eine Abfrage der Maske erscheinen soll. Die eigentliche Verschlagwortung wird dann anschließend im Archiv über ein Formular durchgeführt.

Diese Anforderung lässt sich relativ einfach umsetzen – wenn man erst mal die passenden Events gefunden hat. Die Verschlagwortungsmaske lässt sich einfach unterdrücken – das passiert über eine Konfigurationseinstellung, die man auch per Skript setzen kann:

```
function eloWorkspaceStarted() {
  archive.setUserOption("EloJ.B.IndexDialogOnDemand", "false");
}
```

Das passende Event zur Ablage ist eloInsertDocumentEnd – die dazu gehörende Start Methode würde zwar noch besser passen, sie erlaubt aber keine Änderung des Sord Eintrags und kann deshalb hier nicht verwendet werden.

Die Auswahl der Maske erfolgt am besten über einen CommandLinkDialog. Nachdem der Anwender die passende Maske ausgewählt hat, wird der aktuelle Archiveintrag entsprechend abgeändert.

```
var maskSelection = ["Rechnung", "Bestellung", "Freie Eingabe"];
var destPath = "¶Persönlicher Bereich¶Archivpostbox";

function eloInsertDocumentEnd(doc) {
  var selection = workspace.showCommandLinkDialog(
    "Verschlagwortungsmaske auswählen",
    "Wählen Sie den passenden Dokumententyp aus.",
    "", CONSTANTS.DIALOG_ICON.QUESTION, maskSelection, [], []
  );

  if (selection > 0) {
    var changedSord = ixc.changeSordMask(doc.sord, maskSelection[selection - 1],
EditInfoC.mbSord).sord;
    doc.setSord(changedSord);

    asyncGotoId(doc.id);
  }
}
```

Die Methode asyncGotoId löst im Prinzip nur ein gotoId aus. Allerdings wird der Aufruf verzögert, damit sich der GotoId Aufruf nicht mit dem Ende des Skript-Events überschneidet. Das kann zu sporadischen Fehlern in der Bearbeitung führen, die nur schwer erkannt und behoben werden können.

```
function asyncGotoId(docid) {
  var runnable = new java.lang.Runnable({
    run: function() { workspace.gotoId(docid); }
  });
  java.awt.EventQueue.invokeLater(runnable);
}
```

Weiterhin wurde in dem Projekt gefordert, dass Postbox-Dokumente einfach in einen speziellen Ordner „Archivpostbox“ im persönlichen Bereich des Anwenders geschrieben werden sollen. Natürlich soll auch für diese Dokumente die Auswahl angezeigt werden. Hierzu wurde der Button „Archivablage“ umdefiniert und mit einer eigenen Funktion versehen.

```
function eloInsertIntoArchiveStart() {
  var id = workspace.userNodeId;
  var dataItem = archive.getElementByArcpathRelative(id, destPath);

  var items = intray.allSelected;
  while (items.hasMoreElements()) {
    var item = items.nextElement();
    item.insertIntoArchive(dataItem.id, "", "");
  }

  return -1;
}
```

Microsoft Office Einbindung

FORMULARFELDER BEIMCHECKOUT FÜLLEN

Dieses Skript zeigt, wie man beim Auschecken eines Dokuments die Formularfelder eines Word-Dokuments aus den Verschlagwortungsdaten füllen kann.

Dazu müssen die Namen der Word-Formularfelder mit dem Gruppennamen in der ELO Maskendefinition identisch sein. Durch dieses Gruppen-Formularfeldnamen Mapping ist keine Konfiguration zusätzlich notwendig.

```
// FillFormFields
// (c) by ELO Digital Office GmbH, Jan 2014
//
// This script fills all Word form fields
// from the indexing data of the item and
// the parent of the item when a new Document
// is created.
//
// The name of the Word form field has to
// be ELO_<group_name> - e.g. the indexing
// line KDNR will be inserted into the Word
// form field ELO_KDNR.
//
// ELO Java Client 9.00.000 or newer required.

log.info("Script started: FillFormField, Version 9.00");



function eloCheckoutDocumentAvailable( id, file ) {
  var ext = utils.getFileExtension(file);
  if (ext && (ext.toLowerCase() == "docx")) {
    fillFormFields.fill(id, file);
  }
}

function FillFormFields() {
}
```

```
var fillFormFields = new FillFormFields();

FillFormFields.prototype.fill = function(id, file) {
  var item = archive.getElement( id );
  if (item.sord.doc == 0) {
    var parent = item.parent;

    this.processWord(parent, item, file);
  }
}

FillFormFields.prototype.processWord = function(parent, item, file) {
  ComThread.InitSTA();
  try {
    var word = new ActiveXComponent("Word.Application");
    Dispatch.put(word, "Visible", false);

    var documents = Dispatch.get(word, "Documents").toDispatch();
    var doc = Dispatch.call(documents, "Open", file.path).toDispatch();

    this.processItem(doc, parent);
    this.processItem(doc, item);

    Dispatch.call(doc, "Save");
    var aw = Dispatch.get(doc, "ActiveWindow").toDispatch();
    Dispatch.call(aw, "Close");
    Dispatch.call(word, "Quit");
  } catch(e) {
    log.info("Error processing FormFields: " + e);
    workspace.showAlertBox("ELO", e);
  } finally {
    ComThread.Release();
  }
```

```
}

FillFormFields.prototype.processItem = function(doc, item) {
  var objKeys = item.sord.objKeys;

  for (var i = 0; i < objKeys.length; i++) {
    var key = objKeys[i];
    var name = "ELO_" + key.name;
    if (key.data && (key.data.length > 0)) {
      var value = key.data[0];
      this.writeProperty(doc, name, value);
    }
  }
}

FillFormFields.prototype.writeProperty = function(doc, propertyName, value) {
  try {
    var obj = Dispatch.call(doc, "FormFields", propertyName).toDispatch();
    Dispatch.put(obj, "Result", value);
  } catch(e) {
    log.debug("Error writing word property " + propertyName + " : " + e);
  }
}
```

FORMULARFELDER AUS EINEM SKRIPT LESEN ODER SCHREIBEN

Diese Libray hilft dabei, Word Dokumente aus Skripten heraus zu bearbeiten.

```
myDoc = new Word();
myDoc.openDocument("c:\\temp\\myDocument.docx", true);
myDoc.saveAsPdf("c:\\temp\\myDocument.pdf");
myDoc.closeDocument(false);
```

Speichern Sie die nachfolgenden Teile als lib_Word ab und laden Sie diese per include in beliebige Java Client Skripte.

```
importPackage(Packages.com.jacob.com);
importPackage(Packages.com.jacob.activeX);
```

```
importPackage(Packages.com.ms.activeX);
importPackage(Packages.com.ms.com);

ComThread.InitSTA();

const lib_msoffice_SaveAsPdf = 17;
const lib_msoffice_IgnoreNotSaved = 0;

function Word(){
}// Öffnet ein vorhandenes Word Dokument zur Bearbeitung
//
// fileName: Dateiname mit Pfad
// isVisible: Word anzeigen oder verbergen
//
Word.prototype.openDocument = function (fileName, isVisible) {
  try {
    this.word = new ActiveXComponent("Word.Application");
    Dispatch.put(this.word, "Visible", isVisible);

    this.documents = Dispatch.get(this.word, "Documents").toDispatch();
    this.doc = Dispatch.call(this.documents, "Open", fileName).toDispatch();
  } catch(e) {
    log.info("100: Error opening document " + fileName + ", reason: " + e);
    throw("100: Error opening Word document");
  }
}

// Schließt das aktuelle Word Dokument. Optional wird
// es vor dem Schließen noch gespeichert. Anderfalls
// werden mögliche Änderungen verworfen.
//
// withSave: Dokument vor dem Schließen speichern
//
Word.prototype.closeDocument = function (withSave) {
  if (!this.doc) {
```

```
    // kein Dokument offen, nichts zu tun.
    return;
  }

  if (withSave) {
    try {
      Dispatch.call(this.doc, "Save");
    } catch(e) {
      log.info("102: Error saving Word document, reson: " + e);
      throw("102: Error saving Word document");
    }
  }

  try {
    var aw = Dispatch.get(this.doc, "ActiveWindow").toDispatch();
    Dispatch.call(aw, "Close", 0);
    // Dispatch.call(this.word, "Quit", lib_msoffice_IgnoreNotSaved);
  } catch(e) {
    log.info("101: Error closing document, reason: " + e);
    throw("101: Error closing Word document: " + e);
  }
}

// Speichert das aktuelle Dokument im PDF Format.
//
// pdfFileName: Dateiname und Pfad der Zieldatei
//
Word.prototype.saveAsPdf = function (pdfFileName) {
  try {
    Dispatch.call(this.doc, "SaveAs", pdfFileName, lib_msoffice_SaveAsPdf);
  } catch(e) {
    log.info("103: Error saving Word document as PDF, reason: " + e);
    throw("103: Error saving PDF document");
  }
}
```

```
// Setzt das "Geschützt" Flag im aktuellen Word Dokument
//
Word.prototype.protect = function () {
  try {
    Dispatch.call(this.doc, "Protect", 2, true, "");
  } catch (e) {
    log.info("104: Cannot protect document, reason: " + e);
    //throw("104: Cannot protect document");
  }
}

// Setzt das "Geschützt" Flag zurück
//
Word.prototype.unprotect = function () {
  try {
    Dispatch.call(this.doc, "Protect", -1, true, "");
  } catch (e) {
    log.info("105: Cannot unprotect document: " + e);
    //throw("105: Cannot unprotect document");
  }
}

Word.prototype.getDocumentProperty = function (name) {
  var properties = Dispatch.get(this.doc, "BuiltInDocumentProperties").toDispatch();
  var prop = Dispatch.call(properties, "Item", name).toDispatch();
  var ty = Dispatch.get(prop, "type").getInt();
  if (ty == 4) {
    var value = Dispatch.get(prop, "Value").getString();
    return value;
  }

  return "";
}
```

```
// Liest ein Word Property Feld aus und gibt den Wert zurück.
//
// propertyName: Name des zu lesenden Felds
//
Word.prototype.getPropertyValue = function (propertyName) {
  try {
    var property = Dispatch.call(this.doc, "FormFields", propertyName).toDispatch();
    var fieldType = Dispatch.get(property, "Type");
    if (fieldType == 71) {
      var checkBox = Dispatch.get(property, "CheckBox").toDispatch();
      var checked = Dispatch.get(checkBox, "Value");
      return checked;
    } else {
      var text = String(Dispatch.get(property, "Result"));
      return text;
    }
  } catch(e) {
    log.info("106: Error reading property " + propertyName + ", reason: " + e);
    throw("106: Error reading property " + propertyName);
  }
}

// Schreibt einen neuen Wert in ein Word Property Feld.
//
// propertyName: Name des zu schreibenden Felds
// newValue: Neuer Inhalt des Feldes
//
Word.prototype.setPropertyValue = function (propertyName, newValue) {
  try {
    var property = Dispatch.call(this.doc, "FormFields", propertyName).toDispatch();
    var fieldType = Dispatch.get(property, "Type");
    if (fieldType == 71) {
      var checkBox = Dispatch.get(property, "CheckBox").toDispatch();
      Dispatch.put(checkBox, "Value", newValue);
```

```
    } else {
      Dispatch.put(property, "Result", newValue);
    }
  } catch(e) {
    log.info("106: Error writing property " + propertyName + ", reason: " + e);
    throw("106: Error writing property " + propertyName);
  }
}

// Kopiert die FormFields aus dem angegebenen Word Dokument in
// das aktuelle Word Dokument. Es kann optional ein Prefix
// angegeben werden, dann werden nur Felder kopiert, die mit
// dieser Zeichenfolge beginnen.
//
// sourceDokument: lib_msoffice Word Dokument mit den Quelldaten
// propertiesPrefix: optionale Einschränkung auf Feldnamen
//
Word.prototype.copyProperties = function (sourceDocument, propertiesPrefix) {
  var ff = Dispatch.get(this.doc, "FormFields").toDispatch();
  var count = Dispatch.get(ff, "Count");

  for (var i = 1; i < count; i++) {
    var field = Dispatch.call(ff, "Item", i).toDispatch();
    var name = Dispatch.get(field, "Name").getString();
    log.info("Field name: " + name);

    if (!propertiesPrefix || name.startsWith(propertiesPrefix)) {
      try {
        var value = sourceDocument.getPropertyValue(name);
        this.setPropertyValue(name, value);
      } catch(e) {
        log.info("107: Cannot copy property " + name);
      }
    }
  }
```

```
}

// Kopiert einen Word Range aus einem Dokument in ein anderes Dokument.
// Der Kopiervorgang erfolgt über das Clipboard und nimmt Formatierungen
// und eingebettete Grafiken mit.
Word.prototype.copyRange = function (docFrom, sourceRangeNumber, destRangeNumber) {
  try {
    var sectionS = Dispatch.call(docFrom.doc, "Sections", sourceRangeNumber).toDispatch();
    var rangeS = Dispatch.get(sectionS, "Range").toDispatch();
    Dispatch.call(rangeS, "Copy");

    var sectionD = Dispatch.call(this.doc, "Sections", destRangeNumber).toDispatch();
    var rangeD = Dispatch.get(sectionD, "Range").toDispatch();
    Dispatch.call(rangeD, "Paste");
  } catch(e) {
    log.info("Error moving range " + sourceRangeNumber + " : " + e);
  }
}

// Kopiert die Verschlagwortung in Word Formularfelder. Es kann
// optinal ein Prefix angegeben werden. In diesem Fall werden
// nur Felder kopiert, die im Word Formular den Namen
// <prefix><ELO Gruppenname> tragen. Andernfalls werden alle
// Felder kopiert, die ein Word Formularfeld mit dem Gruppennamen
// besitzen.
//
// indexing: Java Client ArchivElement Objekt oder IX Sord Objekt
// propertiesPrefix: optionaler Prefix für die Word Felder
//
Word.prototype.copyIndexToWord = function (indexing, propertiesPrefix) {
  var sord = indexing.sord;
  if (!sord) { sord = indexing };

  var objKeys = sord.objKeys;
```

```
  for (var k = 0; k < objKeys.length; k++) {
    var key = objKeys[k];
    var value = (key.data && key.data.length > 0) ? key.data[0] : "";
    var name = key.name;
    if (propertiesPrefix) {
      name = propertiesPrefix + name;
    }

    try {
      this.setPropertyValue(name, value);
    } catch(e) {
      log.debug("Field not available: " + name);
    }
  }
}

// Kopiert die Formularfelder eines Word Dokuments in die
// Verschlagwortung eines Java Client ArchiveElement Objekts.
// Es kann optional ein Prefix angegeben werden, dann werden
// nur die Felder kopiert, die mit dem Prefix beginnen und
// anschließend den Gruppennamen besitzen.
//
// indexing: Java Client ArchivElement Objekt oder IX Sord Objekt
// propertiesPrefix: optionaler Prefix für die Word Felder
//
Word.prototype.copyWordToIndex = function (indexing, propertiesPrefix) {
  var sord = indexing.sord;
  if (!sord) { sord = indexing };

  var objKeys = sord.objKeys;
  var found = false;

  for (var k = 0; k < objKeys.length; k++) {
    var key = objKeys[k];
    var name = key.name;
```

```
    if (propertiesPrefix) {
      name = propertiesPrefix + name;
    }

    try {
      var value = this.getPropertyValue(name);
      key.data = [value];
      found = true;
    } catch(e) {
      log.debug("Field not available: " + name);
    }
  }

  if (found && indexing.sord) {
    indexing.sord = sord;
  }
}
```

AUTOMATISCHER CHECKIN BEIM SCHLIEßEN VON WORD

Wenn ein Anwender ein Dokument bearbeiten will, muss er es auschecken. Es wird dann als gesperrt markiert und eine Kopie der Datei im CheckOut Verzeichnis gespeichert. Dort kann es beliebig lange bearbeitet werden. Wenn der Bearbeiter fertig ist, löst er über die Funktion CheckIn eine neue Dokumentenversion aus und die Sperre wird zurückgenommen.

Wenn man nur kleine Dokumente, wie z.B. eine Telefonnotiz, erzeugen möchte, ist dieses explizite CheckIn unnötig umständlich. Wenn das Programm, welches die Bearbeitung durchführt, die Datei während der kompletten Bearbeitungszeit geöffnet hält, kann man auch die Auto-CheckIn Funktion des Java Clients einsetzen, um ein Dokument aus einer Vorlage zur erzeugen und nach Abschluss der Arbeiten automatisch einzuchecken.

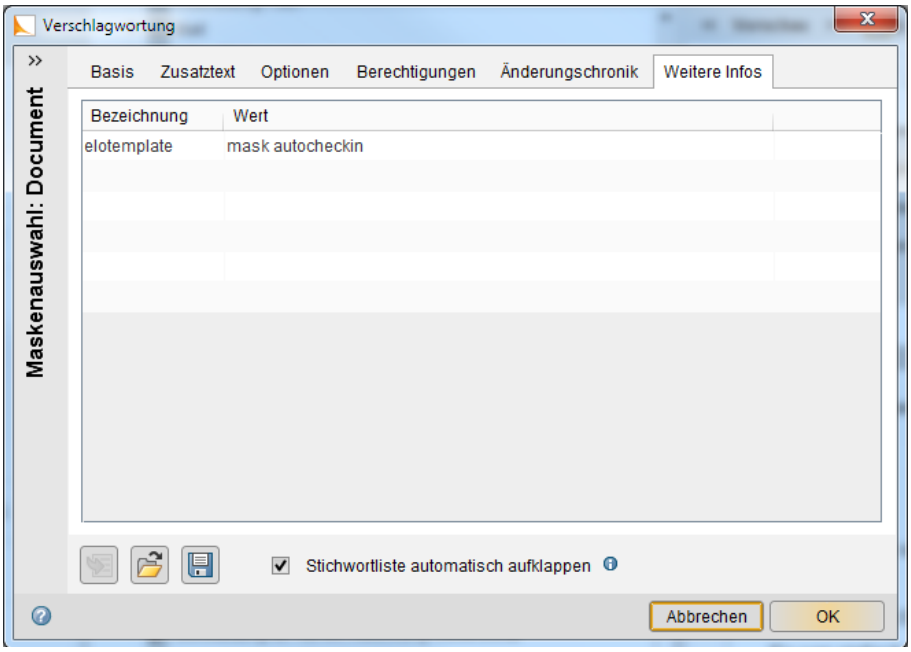


Abbildung 48: Vorlage mit Markierung zum Auto-Checkin und Maskenübernahme

In diesem Fall muss man nur in den „Weitere Infos“ des jeweiligen Vorlagendokuments einen Eintrag mit dem Namen „elotemplate“ anlegen und als Wert „autocheckin“ eintragen. Wenn der Anwender so ein neues Dokument aus einer Vorlage erzeugt, wird eine Dateikopie im CheckOut Verzeichnis angelegt und die Anwendung, z.B. Word, geöffnet. Der Java Client prüft nun regelmäßig im Hintergrund ab, ob er einen exklusiven Zugriff auf die Datei bekommen kann. Wenn das Dokument unter Word gespeichert und geschlossen wurde, ist das der Fall und der Java Client führt automatisch ein CheckIn durch.

Es gibt noch eine zweite Option „mask“ – über diese kann man einstellen, dass das neue Dokument nicht die Standardverschlagwortungsmaske für neue Dokumente erhält, sondern die Maske und Indexzeileninhalte des Vorlagendokuments kopiert.

NEUES DOKUMENT AUS EXTERNER QUELLE ERZEUGEN

Über die Funktion „Neues Dokument aus Vorlage“ kann ein Anwender eine Kopie eines Vorlagendokuments im CheckOut Bereich zur Bearbeitung erzeugen. Wenn das Dokument aber nicht aus einer Vorlage, sondern aus einer externen Quelle kommt, kann man etwas Ähnliches auch per Skript machen.

```
function createWorkItem(parent, maskName, itemName, sourceFile) {
var doc = ixc.createDoc( parent.id, maskName, null, EditInfoC.mbSord );
var sord = doc.sord;
sord.lockId = workspace.userId;
sord.name = itemName;
var newElem = parent.addStructure(sord);
newElem.setLock();
sord.id = newElem.id;
```

```
var destFile = checkout.addDocument(sord, sourceFile);
Desktop.getDesktop().edit(destFile);
}
```

Es wird zuerst mittels createDoc ein Sord Objekt für das neue Dokument erzeugt und dort eine Sperre auf den aktuellen Anwender sowie die Kurzbezeichnung eingetragen. Anschließend wird addStructure und nicht addDocument aufgerufen, da die Dokumentendatei noch gar nicht eingefügt werden soll. Zuletzt wird die externe Datei in den CheckOut Bereich kopiert und zur Bearbeitung aktiviert.

EXCEL DOKUMENT AUS VERSCHLAGWORTUNG FÜLLEN

Über die COM Schnittstelle kann man relativ einfach Excel Dokumente mit Daten aus der Verschlagwortung füllen. Das Beispiel läuft über alle selektierten Dokumente im aktuellen View (z.B. einer Trefferliste) und schreibt die Kurzbezeichnung, den Maskennamen und alle Indexfelder in ein Excel Dokument, welches vorher aus dem Dateisystem geladen wurde.

```
function storeItems() {
  var excel = new Excel();
  excel.openDocument("d:\\temp\\Trefferliste.xlsx", true);
  var items = workspace.activeView.allSelected;

  var line = 3;
  while (items.hasMoreElements()) {
    var item = items.nextElement();
    excel.setCell(line, 2, item.name);
    excel.setCell(line, 3, item.sord.maskName);

    var keys = item.sord.objKeys;
    for (var k = 0; k < keys.length; k++) {
      var data = keys[k].data;
      if (data && (data.length > 0)) {
        excel.setCell(line, 4 + k, data[0]);
      }
    }

    line++;
  }
}
```

Dieses Skript verwendet eine Wrapper-Klasse, die wir in verschiedenen Projekten zur vereinfachten Ansprache der Excel COM Schnittstelle über Jacob geschrieben haben.

```
// Konstruktor
function Excel() {
  this.document = null;
}

// Öffnet ein vorhandenes Excel Dokument zur Bearbeitung
//
// fileName: Dateiname mit Pfad
// isVisible: Excel anzeigen oder verbergen
//
Excel.prototype.openDocument = function (fileName, isVisible) {
  try {
    this.excel = new ActiveXComponent("Excel.Application");
    Dispatch.put(this.excel, "Visible", isVisible);
    Dispatch.put(this.excel, "DisplayAlerts", false);

    this.workbooks = Dispatch.get(this.excel, "Workbooks").toDispatch();
    this.document = Dispatch.call(this.workbooks, "Open", fileName).toDispatch();
    this.sheet = Dispatch.call(this.document, "Sheets", 1).toDispatch();
  } catch(e) {
    log.info("108: Error opening Excel document " + fileName + ", reason: " + e);
    throw("108: Error opening Excel document");
  }
}

// Schließt das aktuelle Excel Dokument. Optional wird
// es vor dem Schließen noch gespeichert. Anderfalls
// werden mögliche Änderungen verworfen.
//
// withSave: Dokument vor dem Schließen speichern
//
Excel.prototype.closeDocument = function (withSave) {
  if (!this.document) {
```

```
    // kein Dokument offen, nichts zu tun.
    return;
  }

  if (withSave) {
    try {
      Dispatch.call(this.document, "Save");
    } catch(e) {
      log.info("109: Error saving Excel document, reson: " + e);
      throw("109: Error saving Excel document");
    }
  }

  try {
    Dispatch.call(this.document, "Close", false);
    this.document.safeRelease();
    Dispatch.call(this.excel, "Quit");
    this.excel.safeRelease();
  } catch(e) {
    log.info("110: Error closing document, reason: " + e);
    throw("110: Error closing Word document.");
  }
}

// Speichert das aktuelle Dokument im PDF Format.
//
// pdfFileName: Dateiname und Pfad der Zieldatei
//
Excel.prototype.saveAsPdf = function (pdfFileName) {
  try {
    Dispatch.call(this.document, "ExportAsFixedFormat", 0, pdfFileName, 0, 1, 1,1, 1, false);
  } catch(e) {
    log.info("111: Error saving Excel document as PDF, reason: " + e);
    throw("111: Error saving PDF document");
  }
```

```
}

// Füllt eine Zelle mit dem angegebenen Wert und Farbe
//
// row : Zeile im Excel Worksheet
// col : Spalte
// value : Einzutragender Wert
// color : Excel Farbindex
//
Excel.prototype.setCell = function (row, col, value, color) {
  var cell = Dispatch.call(this.sheet, "Cells", row, col).toDispatch();
  Dispatch.put(cell, "Value", String(value));
  if (color) {
    var interior = Dispatch.get(cell, "Interior").toDispatch();
    Dispatch.put(interior, "ColorIndex", color);
  }
}
```

	A	B	C	D	E	F	G	H
1								
2								
3		Musterrechnung_Einfach_Basic6089296329655491076	Rechnung		Lieferant GmbH	198	235,62	37,62
4		Beispielrechnung_Handel_Warenrechnung4540800616529401493	Rechnung		MUSTERLIEFERANT GMBH	448,99	518,99	70
5		Musterrechnung_Versicherung_Haftpflicht8335976420164004361 geändert	Rechnung		MVM Musterhafter Versicherungsverein	50	59	9
6		Musterrechnung_Versicherung_Haftpflicht8335976420164004361 geändert	Rechnung		MVM Musterhafter Versicherungsverein	50	59	9
7		Musterrechnung_Versicherung_Haftpflicht8335976420164004361 geändert	Rechnung		MVM Musterhafter Versicherungsverein	50	59	9
8		18: Beispielrechnung_Handel_Warenrechnung4540800616529401493	Rechnung	908711	MUSTERLIEFERANT GMBH	448	11518	70
9		Musterrechnung_Versicherung_Haftpflicht8335976420164004361 geändert	Rechnung		MVM Musterhafter Versicherungsverein	5000	5950	950

Abbildung 49: Erzeugtes Excel Dokument

HÄNGENDE EXCEL VERSIONEN

Excel erweckt zwar den Eindruck, als könnte man mehrere Instanzen parallel starten und dann unabhängig voneinander arbeiten. In der Praxis ist das aber nur eingeschränkt der Fall. Wenn eine Excel Instanz in einem modalen Zustand ist, blockiert das durchaus auch die anderen Instanzen. Schlimmer wird es noch dadurch, dass auch der Microsoft Excel Preview Handler nur eine weitere Excel Ansicht ist. Das führt dazu, dass sich der Java Client beim Blättern durch Excel Dokumente über den Preview „aufhängen“ kann. Tatsächlich hängt nicht der Java Client, sondern nur die Excel Instanz, die für den Preview verwendet wird. Aus Anwendersicht ist aber der Client der Schuldige.

Auflösen kann man das Problem, indem man die hängende Excel Instanz über den Windows TaskManager beendet. Allerdings kann man nicht sehen, welche der vorhandenen Excel Versionen für die Blockade zuständig ist. Man kann einfach nur auf gut Glück eine nach der anderen stoppen und sehen, ob der Client wieder frei ist.

Eine etwas brutale Version dieser Vorgehensweise ist in der folgenden Batch Datei implementiert: Es werden einfach alle Excel Instanzen beendet. Diese Datei auf dem Desktop kann mit einem Doppelklick die Excel Verklemmung lösen.

```
@echo off

echo .
echo Dieser Aufruf beendet alle laufenden Excel Instanzen.
echo Veränderte Inhalte werden nicht gespeichert.
echo .
echo Wollen Sie das wirklich? (J)
echo .

set QUERY=
set /P QUERY=Geben Sie J oder N ein: %=%

if /I "%QUERY%"=="J" goto yes
if /I "%QUERY%"=="JA" goto yes

goto no

:yes
taskkill /F /IM excel.exe

:no
```

Wie immer im Leben: Man bezahlt einen Preis dafür. Mit der Ausführung der Batch Datei werden alle Excel Fenster gekillt. Ohne zu speichern! Alle Eingaben seit dem letzten Speichervorgang gehen verloren.

ELO Automation Services

MICROSOFT OFFICE DOKUMENT IN PDF UMWANDELN

Der ELOas besitzt im Modul fu eine Methode die ein Microsoft Office Dokument (doc, docx, xls, xlsx, ppt, pptx) in das PDF Format umwandelt und dieses als neue Version abspeichert.

```
<script>
  fu.convertAsNewVersion(Sord.id);
</script>
```

Wenn man keine neue Version erstellen möchte, sondern irgendeine andere Aktion durchführen möchte, kann man die Methode fu.convertToPdf(sourceName, destinationName) verwenden.

FEED BEITRAG AUS ELOAS ERSTELLEN

Der Dokumentenfeed ist sowohl für die Kommunikation der Anwender untereinander wie auch dem Informationstransfer vom System zum Anwender gedacht. Aus diesem Grund kann es sinnvoll sein, im Prozess beim Erreichen bestimmter Milestones automatisiert einen entsprechenden Feed Eintrag zu erstellen.

Das Modul ix stellt dafür eine einfache Funktion zur Verfügung:

```
<script>
  var milestoneInfo = "Der Prozesszustand ‘Abgeschlossen’ wurde erreicht.”;
  ix.addFeedComment( Sord.guid, 0, milestoneInfo );
</script>
```

INDEXINFORMATION AUS DEM VOLLTEXT GEWINNEN

Bestimmte Indexbegriffe mit einem festen formalen Aufbau lassen sich gut aus dem Volltext gewinnen. Hierzu muss man in einer ELOas Regel nur den Volltext als String vom Indexserver auslesen und über einen regulären Ausdruck nach dem Begriff suchen.

```
<script>
  var text = String(ix.getFulltext(Sord.id));
  if (text) {
    log.info(text);
    var id = text.match(/TTS[0-9]{6}/);
    if (id) {
      log.info(id);
      BEMERKUNG = id;
```

```
      EM_WRITE_CHANGED = true;
    }
  }
</script>
```

Das Schreiben des gefundenen Ausdrucks muss so erfolgen, dass das Dokument in Zukunft nicht mehr in der Suchliste auftritt, andernfalls wird das Dokument immer- und immer wieder bearbeitet werden. Zudem sollte man dafür sorgen, dass bei älteren Dokumenten nicht beliebig lang auf den Volltext gewartet wird – falls zu einem Dokument kein Volltext ermittelt werden konnte.

RÜCKGABEWERTE AUS EINER DIREKT RULE

Eine DIRECT Rule kann einen String als Rückgabewert setzen, der dann auf aufrufenden Programm übergeben wird. Wenn man viele Daten zurückgeben möchte, wäre es praktischer, wenn man ein komplettes JavaScript Objekt liefern könnte. Hierfür bietet sich der Einsatz der verfügbaren JSON Library an. Mit ihrer Hilfe kann man über die Funktion stringify aus einem JavaScript Objekt einen JSON String erzeugen. Dieser kann auf der Empfängerseite dann leicht wieder in ein JavaScript Objekt zurückgewandelt werden.

```
<script>
  var result = [
    { v: 'abc', description: '123' },
    { v: 'def', description: '145' },
    { v: 'ghi', description: '678' }
  ];

  elo.setAnswer(JSON.stringify(result));
</script>
```

WORKFLOW AN MEHRERE NACHFOLGER WEITERLEITEN

Ein ELOas Workflow-Knoten wird am Ende der Bearbeitung durch Setzen der Variablen EM_WF_NEXT weitergeleitet. In vielen Fällen gibt es zum ELOas Knoten nur einen Nachfolger, deshalb sieht man oft den Befehl EM_WF_NEXT = "0";

Das ist aber nur eine Verkürzung der verschiedenen Möglichkeiten bei der Weiterleitung. In dem einfachen Fall wird der Workflow an den ersten Nachfolgerknoten (Index 0 in der Nachfolgerliste) weitergeleitet. Man kann aber auch eine Liste von Nachfolgern bestimmen. In diesem Fall werden die Indexnummern jeweils mit einem Pilcrow getrennt in die Liste eingetragen.

Das folgende Beispiel prüft nach, ob der Workflow-Knotenname CompareELOOUTL12 lautet und ob der Wert in der Indexzeile ELOOUTL1 identisch zu dem Wert in der Indexzeile ELOOUTL2 ist. In diesem Fall wird an den ersten und zweiten Nachfolger weitergeleitet. Andernfalls wird nur an den ersten Nachfolger weitergeleitet. Solche Unterscheidungen hat man z.B. in Rechnungsworkflows. Wenn der Rechnungsbetrag einen bestimmten Wert nicht überschreitet, wird nur an den Sachbearbeiter weitergeleitet, andernfalls an den Sachbearbeiter und an den Vorgesetzten.

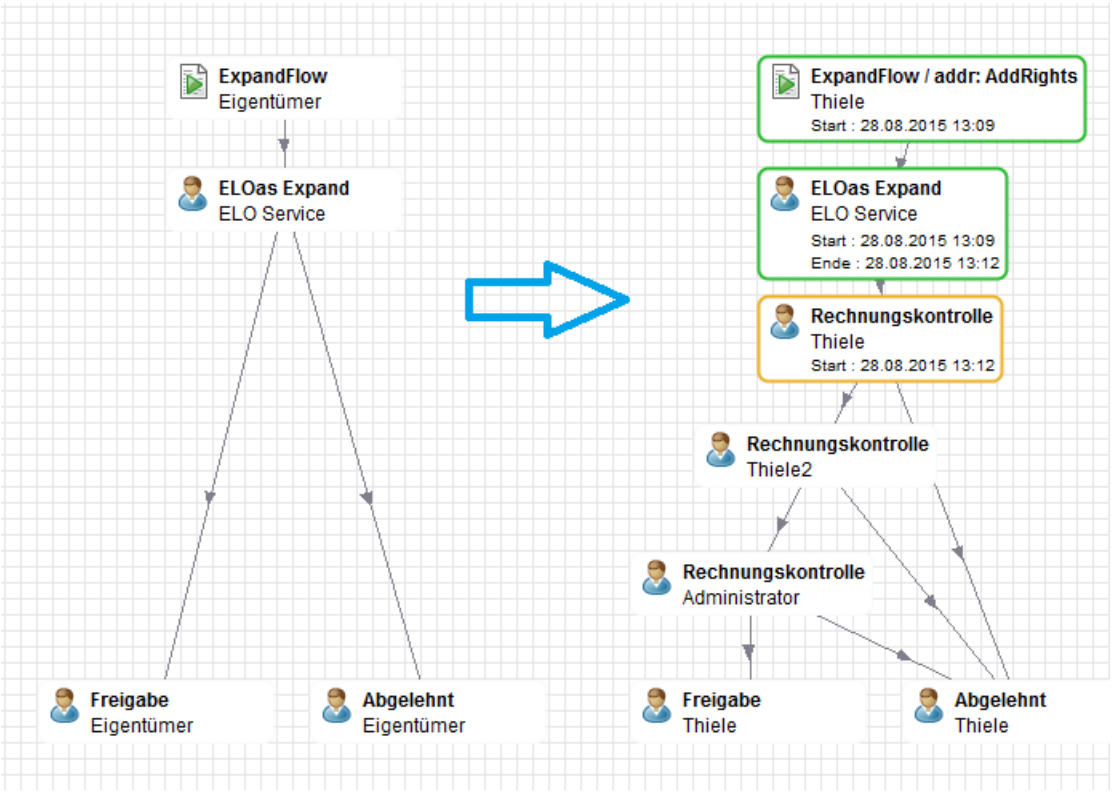
```
<script>
if (EM_WF_NODE.nodeName == "CompareELOOUTL12") {
  if (ELOOUTL1 == ELOOUTL2) {
    EM_WF_NEXT = "0¶1";
  } else {
    EM_WF_NEXT = "0";
  }
}
</script>
```

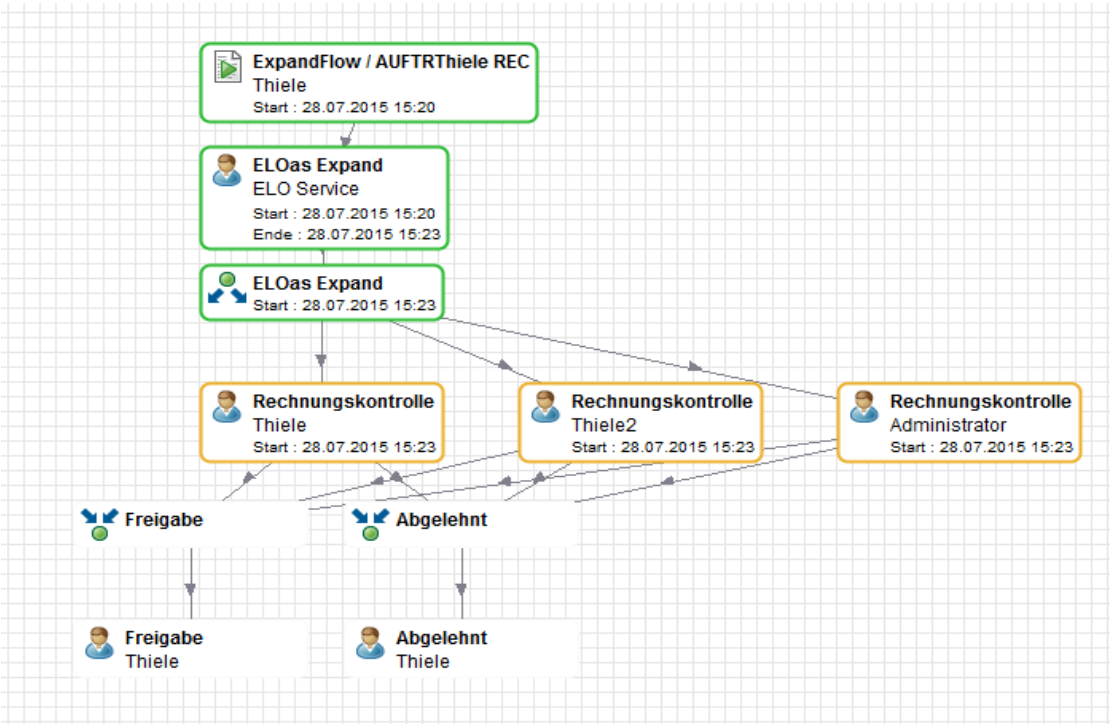
NACHFOLGERLISTE DYNAMISCH ERWEITERN

Es kommt in der Praxis häufig vor, dass sich erst zur Laufzeit des Workflows entscheidet, wer alles an einer bestimmten Entscheidung teilnimmt. In diesem Fall kann man die Teilnehmerliste nicht statisch im Workflowdesigner festlegen, da zu diesem Zeitpunkt weder die Namen noch die genaue Anzahl der Teilnehmer festliegt.

Für diesen Fall gibt es im ELOas Modul wf die Funktionen wf.expandNodeParallel und wf.expandNodeLinear. Im Designer wird nur ein einzelner Knoten für den Verteiler eingefügt und dem ELOas zugeordnet. Wenn so ein Knoten im ELOas aktiv wird, prüft das Skript ab, ob es sich um einen expandieren Knoten handelt (das kann z.B. anhand des Knotennamens erfolgen). Wenn ja, wird die Anwenderliste ermittelt, z.B. aus einer Indexzeile, die im Laufe des Prozesses mit den Namen der betroffenen Personen gefüllt wurde und anschließend wird die expandNode Methode aufgerufen. Zum Abschluss wird der Workflow weitergeleitet.

```
<script>
  log.debug("Process Rule expand.");
  var names =  ["Thiele", "Thiele2", "Administrator"];
  wf.expandNodeLinear( EM_WF_NODE.flowId, EM_WF_NODE.nodeId,
                                names, "Rechnungskontrolle" );
  EM_WF_NEXT = "0";
</script>
```


ExpandFlow
Eigentümer
ELOas Expand
ELO Service
Freigabe
Eigentümer
Abgelehnt
Eigentümer
ExpandFlow / addr: AddRights
Thiele
Start : 28.08.2015 13:09
ELOas Expand
ELO Service
Start : 28.08.2015 13:09
Ende : 28.08.2015 13:12
Rechnungskontrolle
Thiele
Start : 28.08.2015 13:12
Rechnungskontrolle
Thiele2
Rechnungskontrolle
Administrator
Freigabe
Thiele
Abgelehnt
Thiele


ExpandFlow / AUFTRThiele REC
Thiele
Start : 28.07.2015 15:20
ELOas Expand
ELO Service
Start : 28.07.2015 15:20
Ende : 28.07.2015 15:23
ELOas Expand
Start : 28.07.2015 15:23
Rechnungskontrolle
Thiele
Start : 28.07.2015 15:23
Rechnungskontrolle
Thiele2
Start : 28.07.2015 15:23
Rechnungskontrolle
Administrator
Start : 28.07.2015 15:23
Freigabe
Abgelehnt
Freigabe
Thiele
Abgelehnt
Thiele

ANWENDER EINES WORKFLOWKNOTENS ÄNDERN

Manchmal entscheidet sich erst zur Laufzeit eines Workflows, welcher Anwender einen bestimmten Knoten bearbeiten muss. Bei einem Rechungsworkflow wird die Buchhaltung den Freigabe-Sachbearbeiter ermitteln und in ein Indexfeld SACHBEARBEITER eintragen. Der eigentliche Freigabeknoten wird im Template dem Eigentümer oder einen Dummy-User zugeordnet.

In dem Workflow ist nach dem Buchhaltungsknoten ein ELOas Knoten aktiv, der den Freigabeknoten sucht, die Indexzeile für den Sachbearbeiter ausliest und diesen Mitarbeiter einträgt. Das ist im ELOas mit einer Zeile machbar.

```
<script>
  wf.changeNodeUser("Freigabe Sachbearbeitung", SACHBEARBEITER);
</script>
```

DATUMSFELD EIN JAHR WEITERZÄHLEN

Eine Anfrage aus dem Forum: Bei einer Vertragsverwaltung soll regelmäßig ein Kündigungsdatum geprüft werden. Wenn es erreicht oder überschritten ist, soll eine Aktion ausgeführt werden und das Datum ein Jahr weitergesetzt werden. Diese Anforderung lässt sich mit dem ELOas leicht realisieren.

Zuerst legt man in der Admin Console einen Ruleset an, der eine Suche nach allen aktiven Vertragsdokumenten durchführt. In der onStart Methode formatiert man aus dem aktuellen Tagesdatum ein ISO Datum, da sich dieses leicht gegen das Kündigungsdatum vergleichen lässt.

```
nowIso = Packages.de.elo.mover.main.helper.DateHelper.isoFromDate( new Date() );
```

In der Rule1 prüft man zuerst nach, ob der Eintrag ein Kündigungsdatum besitzt und ob es überschritten ist. Wenn ja, führt man die Aktion aus. Anschließend wird das Jahr aus dem Kündigungsdatum herausgezogen und um eins hochgezählt.

```
if ((nowIso > TERMINATE) && (TERMINATE.length > 4)) {
  // hier die Aktion durchführen
  var year = TERMINATE.substring(0, 4);
  year = parseInt(year, 10) + 1;
  TERMINATE = year + TERMINATE.substring(4);
  EM_WRITE_CHANGED = true;
}
```

In dieser einfachen Form wird nicht geprüft, ob das Datum gültig ist. Aus einem 29.02.2016 wird ein 29.02.2017 – diesen Tag gibt es aber nicht. Das führt hier zwar nicht zu Störungen, möglicherweise aber an anderer Stelle.

NACH MEHREREN INDEXBEGRIFFEN SUCHEN

Prinzipiell führt der ELOas eine Suche nach einem Wert in einer Indexzeile durch. Da es aber häufiger mal vorkommt, dass sich erst aus einer Kombination mehrerer Indexzeilen eine sinnvolle Selektion ergibt, kann man optional auch eine Liste von Indexzeilennamen und Suchwerten eingeben. Die Werte müssen jeweils durch ein Pilcrow Zeichen getrennt geschrieben werden. Und die Zahl der Indexzeilennamen muss identisch zur Zahl der Werte sein.

EIGENE SUCHE AUSFÜHREN

Der ELOas arbeitet im Prinzip eine vorgegebene Suche zu einer Maske und einer oder mehreren Indexzeilen ab. Manchmal muss man aber eine ganz spezielle Suche mit weiteren Einschränkungen oder Optionseinstellungen vornehmen. In diesem Fall kann man ein eigenes FindInfo Objekt erstellen und in der Variablen EM_FIND_INFO zur Verarbeitung anmelden. Das Füllen dieser Variablen muss in dem onStart Event erfolgen, es wird vor der Suche ausgeführt. Bei der Suche wird zuerst geprüft, ob die Variable EM_FIND_INFO mit einem FindInfo Objekt gefüllt ist. Wenn nicht, wird ein FindInfo Objekt aus den Ruleset Parametern erzeugt.

Im Beispiel werden alle Einträge mit einer Haftnotiz mit dem Text „TEST“ gesucht.

```
var findInfo = new FindInfo();
var fbn = new FindByNotes();
fbn.term = “TEST”;
findInfo.findByNotes = fbn;

EM_FIND_INFO = findInfo;
```

FREMDE WEB SEITENINHALTE AUSLESEN

Der ELOas stellt in der Library www eine Methode get zur Verfügung mittels der Sie leicht den Inhalt einer anderen Web Seite als String auslesen können.

```
<script>
  var elo = www.get("https://www.elo.com/wcm/de/produkte/eloprofessional");
  log.info(elo);
</script>
```

Aus dem empfangenen String kann man dann z.B. über reguläre Ausdrücke die gewünschten Informationen auslesen – z.B. Wechselkurse oder Börsendaten.

LESEN UND SCHREIBEN VON MAP FELDERN

Map-Felder liegen parallel zum Sord Objekt in einer eigenen Struktur. Da sie sehr umfangreich sein können, werden sie vom ELOas nicht automatisch mit eingelesen. Sie müssen bei Bedarf per Skript vom Indexserver gelesen werden und nach einer Veränderung auch selber wieder geschrieben werden.

Im Folgenden wird eine kleine Library vorgestellt, welche Herr Weiler für diesen Zweck geschrieben hat:

```
// var sordMap = new SordMap(EM_ACT_SORD.id);
// sordMap.addKeyValue("VAR1", "value1");
// sordMap.addKeyValue("VAR2", "value2");
// sordMap.addKeyValue("VAR3", "value3");
// sordMap.write();

// sordMap = new SordMap(EM_ACT_SORD.id);
// sordMap.read();
// log.debug("VAR2=" + sordMap.getValue("VAR2"));
// log.debug("VAR5=" + sordMap.getValue("VAR5"));

function SordMap(sordId) {

  this.sordId = sordId;
  this.map = [];
}

SordMap.prototype.addKeyValue = function (key, value) {

  this.map.push(new KeyValue(key, value));
};

SordMap.prototype.write = function () {

  ixConnect.ix().checkinMap(MapDomainC.DOMAIN_SORD, this.sordId, this.sordId, this.map,
LockC.NO);
};

SordMap.prototype.read = function () {
```

```
  var data = {};
  var items = ixConnect.ix().checkoutMap(MapDomainC.DOMAIN_SORD, this.sordId, null,
LockC.NO).items;
  items.forEach(function (item) { data[item.key] = item.value; });
  this.data = data;
};

SordMap.prototype.getValue = function (key) {

if (this.data[key]) {
    return this.data[key];
  }
  return "";
};
```

VERSCHLÜSSELTE PASSWÖRTER ENTSCHLÜSSELN

In der ELO Konfiguration können Passwörter verschlüsselt hinterlegt werden. Es gibt ein kleines Tool, welches die Eingabe des Passworts erlaubt und die verschlüsselte Version ausgibt – diese wird dann in die Config-Datei eingetragen.

Wenn ein Skript Zugriff auf das entschlüsselte Passwort benötigt – oder die ELO Verschlüsselung auch für eigene Passwörter verwenden möchte, kann man das so machen:

```
  var password = "52-247-139-10-8-11-59-34";    // crypted password

  try{
    var des = new Packages.de.elo.utils.sec.DesEncryption();
    password = des.decryptIfEncrypted(password);
  } catch (desEx){
    log.info("Cannot decypt password");
  }
```

ELO DOKUMENTENTYP AUS DER DATEI-EXTENSION ERMITTELN

Wenn man über den Client eine Datei ablegt, ermittelt der Client den passenden ELO Typ aus der Dateiextension. Legt man ein Dokument über den ELOas ab, ist das die Aufgabe des ELOas Skripts. Das Modul ix stellt hierfür aber eine passende Methode bereit.

```
log.debug(ix.lookupDocType("test.docx"));
```

DOKUMENT AUF EINEN ANDEREN DATEIPFAD UMZIEHEN

Sie können Dokumente zwar nicht in einen beliebigen Dateipfad legen – aber Sie können zumindest zwischen den definierten Dateipfaden umziehen. Es wäre z.B. denkbar, dass Dokumente nach einem bestimmten Kriterium auf ein anderes physikalisches Medium ausgelagert werden sollen. In diesem Fall erzeugen Sie einen neuen Dateipfad in der Admin Console.

Anschließend erzeugen Sie einen Ruleset, der eine Suche nach dem Kriterium durchführt – z.B. ARCHIVSTATUS = „Zur Archivierung vorgesehen“. Diesen Ruleset können Sie über die Intervallsteuerung so anlegen, dass er nur nachts oder nur am Wochenende läuft, damit er bei großen Dokumentenmengen nicht den normalen Betrieb stört.

In der Rule müssen Sie nun zwei Aktionen durchführen:

1. den Dateipfad auf das andere Medium legen (PathId in ARCHIVE_PATH).
2. den Archivstatus auf „Archiviert“ umstellen, damit der Eintrag aus der Suche entfernt wird.

```
ix.moveToPath(Sord.id,  ARCHIVE_PATH);
ARCHIVSTATUS = "Archiviert";
```

SUCHE NACH ZEITSTEMPEL

Die ELO Datenbank führt einen Zeitstempel mit der letzten Bearbeitung eines Eintrags mit. Das ist primär für die Replikation gedacht, kann aber auch im ELOas verwendet werden.

Wenn man z.B. regelmäßig über die Einträge eines bestimmten Dokumententyps eine Plausibilitätskontrolle durchführen möchte, müsste man theoretisch jeden Schreibzugriff abfangen und die Prüfung durchführen. Einfacher geht es, wenn man es asynchron im ELOas prüft.

Zur Prüfung legt man sich einen Ruleset für diese Maske an. Den <search> Abschnitt muss man manuell bearbeiten, da die GUI keine Einstellungen für eine Zeitstempelsuche vorsieht

```
<search>
  <name>"ELOTIMESTAMP"</name>
  <value>"2013.07.17.00.00.00...2013.07.17.23.59.59"</value>
  <mask>0</mask>
  <max>200</max>
</search>
```

In der Praxis wird man natürlich nicht einen festen Zeitbereich haben, sondern ein gleitendes Fenster, z.B. von „vor 24 Stunden bis jetzt“. In diesem Fall muss man den Wert im <value> Tag per Skript setzen. Der passende Ort dafür ist das onStart Event – es wird vor der Suche ausgeführt.

```
var now = new Date();
nowTS = elo.timeStamp ( now );
now.setDate( now.getDate() – 1 );
yesterdayTS = elo.timeStamp ( now );
EM_SEARCHVALUE = yesterdayTS + “...” + nowTS;
```

WASSERZEICHEN IN PDF DOKUMENT EINFÜGEN

Der ELOas verfügt über eine Reihe von Helper-Funktionen zum Einfügen von Texten und Bildern in PDF und Tiff Dateien. Als Beispiel soll hier gezeigt werden, wie man ein Wasserzeichen in ein PDF Dokument einfügt.

Dafür wird zuerst die Dokumentendatei in den Temp-Ordner geladen, Anschließend wird die Funktion insertTextInPdf aufgerufen. Diese besitzt als Parameter den Text und die Position sowie die Textfarbe mit Transparenz. Zuletzt wird die veränderte Pdf Datei als neue Version eingecheckt und die temporäre Datei gelöscht.

```
var file = fu.getTempFile(Sord.id);
PdfFileHelper.insertTextInPdf("Vertraulich", file, 3, 200, 50, 150, 255, 0, 0, 0.5, 50);
fu.addVersion(Sord.id, file);
fu.deleteFile(file);
```

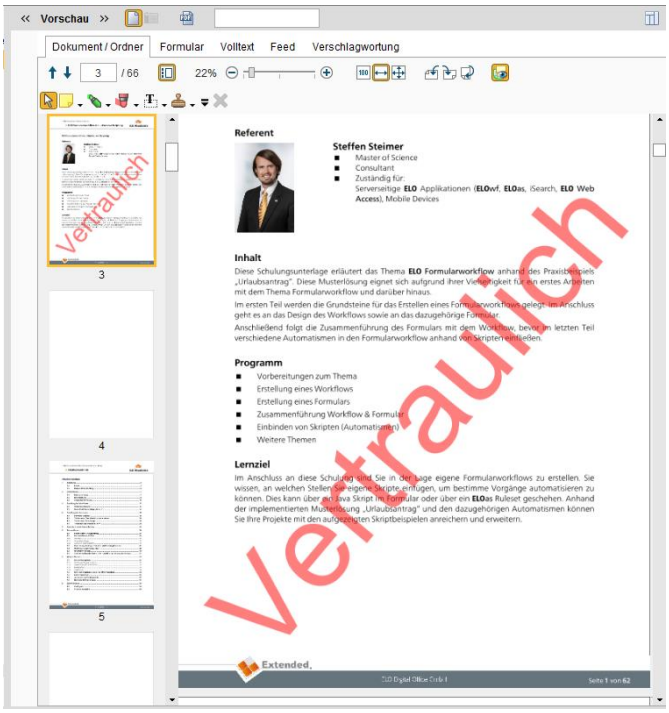


Abbildung 50: Pdf Dokument mit Wasserzeichen

BEARBEITUNGSREIHENFOLGE IM TREEWALK

Bei der Abarbeitung eines TreeWalks im ELOas wird jeder Knoten zweimal besucht. Einmal beim „Abstieg“ – d.h. es wird erst der Ordner aufgerufen, danach alle seine Child-Einträge. Und zum zweiten Mal auf dem „Rückweg“ – nachdem alle Child Einträge (und die Child-Einträge der Children) abgearbeitet wurden, wird er Ordner erneut aufgerufen.

Um zwischen den beiden Aufrufen unterscheiden zu können, wird die globale Variable EM_TREE_STATE gesetzt. Beim Abstieg auf 0 und auf dem Rückweg auf 1. Eine Speicherung von veränderten Werten erfolgt nur auf dem Rückweg. Deshalb sollte man bei der Bearbeitung prinzipiell auf EM_TREE_STATE == 1 prüfen.

```
log.info("# Benenne Dokument um");
if (EM_TREE_STATE == 1 && NAME == "Wait for Fulltext") {
  log.info("################# 1b " + NAME);
  NAME = NOPROJECT;
  EM_WRITE_CHANGED = true;
  log.info("# Dokument erfolgreich umbenannt");
}
```

MASKENÜBERGREIFENDE RULESETS

Im Normalfall führt der ELOas eine Suche nach einer bestimmten Maske und Suchbegriff aus und bearbeitet dann die Trefferliste – in der alle Einträge die gleiche Maske haben. Wenn man einen TreeWalk oder eine Workflow Suche durchführt, kann man aber Treffer unterschiedlicher Masken haben. Das Gleiche passiert, wenn man für die Suche eine reine Suchmaske verwendet.

Damit die „nicht passenden" Einträge korrekt bearbeitet werden können, muss man die globale Variable EM_MASK_LOADED abfragen. Wenn der Wert auf true steht, wurde das Sord Objekt ausgelesen und in die vordefinierten Variablen eingefüllt (z.B. NAME, ELOOUTL1, KDNR…). Das passiert aber nur dann, wenn das Sord Objekt entweder der Suchmaske oder einer der weiteren Zielmasken entspricht. Andernfalls ist EM_MASK_LOADED false und in diesem Fall müssen Sie das Sord Objekt direkt bearbeiten.

Beim Speichern ist die Situation leider genauso kompliziert. Die Basisdaten werden immer aus den globalen Variablen zurückgeschrieben. Diese werden aber auch immer beim Lesen gefüllt. Die Indexzeilen werden nur dann aus den globalen Variablen zurückgeschrieben, wenn es eine passende Maske gibt. Das würde dazu führen, dass man bei passenden Masken die Indexzeilen in die globalen Variablen eintragen muss und andernfalls direkt in das Sord Objekt eintragen muss. Diesen Aufwand kann man vermeiden, indem man die Variable EM_MASK_LOADED auf -1 setzt. Dann werden die Indexwerte nur aus dem Sord Objekt zurückgeschrieben und die globalen Variablen ignoriert.

Die Basisdaten (Kurzbezeichnung, Datum…) werden IMMER in die globalen Variablen eingefüllt und immer von dort zurückgeschrieben, da diese von der Maske unabhängig sind.

Bei Workflows gibt es noch eine Sonderbehandlung. Normalerweise werden Workfloweinträge in einem Ruleset nur bearbeitet, wenn die Maske zu der eingestellten Suchmaske im Ruleset identisch ist. Da Workflows im Allgemeinen Maskenbezogen sind, ist diese Voreinstellung durchaus sinnvoll. Manchmal hat man aber Aufgaben, die über unterschiedliche Masken laufen sollen. In diesem Fall muss man in dem <onstart> Event mitteilen, dass man alle Workflowtermine bearbeiten möchte.

```
<onstart>
  EM_ALLOWALLMASKS = true;
</onstart>
```

AKTIVE WORKFLOW-TERMINE ZU GELÖSCHTEN DOKUMENTEN

Im Standard gibt der Indexserver keine Workflowtermine zu Dokumenten aus, die zwischenzeitlich gelöscht wurden. Das kann aber dazu führen, dass sich das System mit unbearbeiteten Workflows füllt. Da diese nicht sichtbar sind, können sie auch nicht abgeschlossen oder weitergeleitet werden.

Über eine Optionseinstellung kann man die Ausgabe aber erzwingen. Hierzu muss im onstart Event nur die Variable EM_WF_WITH_DELETED auf true gesetzt werden.

```
<onstart>
  EM_WF_WITH_DELETED = true;
```

```
  </onstart>
```

In diesem Fall sollte man aber vorsichtig mit den ausgeführten Aktionen sein – nicht alle Aktionen sind auf gelöschten Einträgen möglich oder erlaubt.

Wenn ein Dokument über den ELOas gelöscht wird, kann man über den Befehl wf. deleteAllWorkflowsOfObject sicherstellen, dass alle Workflows zu dem Eintrag gelöscht werden.

```
wf.deleteAllWorkflowsOfObject(Sord.id);
```

[10] BEARBEITUNGSLISTE AUS EINER REGISTRIERTEN INDEXSERVER-FUNKTION

Der Indexserver bietet über die registered functions die Möglichkeit, vordefinierte Skripte innerhalb des IX auszuführen. Diese Funktion kann der ELOas zur Ermittlung der Bearbeitungsliste verwenden.

Zur Verwendung muss im Indexserver ein Skript mit der registrierten Funktion vorliegen. Solche Skriptfunktionen beginnen mit dem Namenspräfix „RF_“ – z.B. RF_selectedCustomers. Eine ELOas Suche über die registrierte Funktion kann einen String als Eingabeparameter definieren (über den EM_SEARCHVALUE). Die Funktion muss als Rückgabewert eine Liste von ELO Objekt-Ids – Kommasepariert und ohne Leerzeichen – liefern.

```
function RF_ selectedCustomers ( ec, args ) {
  log.info("Start RF_ selectedCustomers : " + args);
  if (args == "NORD") {
    return "3581501,3453004";
  } else {
    return "3581489";
  }
}
```

Im ELOas Ruleset muss nur als EM_SEARCHNAME der Name der registered function eingegeben werden und optional als EM_SEARCHVALUE der Parameter. Dieser kann natürlich auch zur Laufzeit noch in dem onstart Event angepasst werden.

```
<ruleset>
  <base>
    <name>processCustomers</name>
    <search>
      <name>"RF_ selectedCustomers"</name>
      <value>"WEST"</value>
      <mask>0</mask>
      <max>10</max>
```

```
    </search>
    <interval>1M</interval>
    <onstart>
    </onstart>
  </base>
<rule>
  <name>Rule1</name>
  <script>
    log.info("process customer: " + Sord.id + " : " + Sord.name);
  </script>
</rule>
<rule>
  <name>Global Error Rule</name>
  <condition>OnError</condition>
  <script></script>
</rule>
</ruleset>
```

[10] ALLE DOKUMENTENTYPEN BEIM TREEWALK BEARBEITEN

Wenn der ELOas einen Eintrag lädt, trägt er den Inhalt der Indexzeilen in vordefinierte globale Variablen mit dem Gruppennamen der Indexzeile ein. Beim Schreiben passiert es umgekehrt – die Indexzeilen des Sord Objekts werden aus den globalen Variablen gefüllt bevor der Eintrag gespeichert wird. Das hat für den Anwender den großen Vorteil, dass er nicht das ObjKeys Array durchsuchen und dort das data Array lesen oder füllen muss, sondern einfach „KDNR = 12345“ schreiben kann.

Bei einer Suche ist das praktisch, da die Suche im Allgemeinen auf einer Maske ausgeführt wird. Es funktioniert auch, wenn der Anwender in der Rule die Maske ändert, da die vordefinierten globalen Variablen für alle angegebenen Masken des Rulesets erzeugt, geschrieben und gelesen werden.

Schwieriger wird es, wenn die Suche über eine Suchmaske läuft oder über einen TreeWalk. In diesen Fällen kann die Trefferliste mehrere Masken beinhalten. Solange es sich nur um einige wenige Masken handelt, kann man mit den zusätzlichen Masken arbeiten. Sowohl beim Lesen wie beim Schreiben schaut der ELOas auf die aktuelle Maske des Sord Objekts und schreibt bzw. liest die dazu passenden globalen Variablen.

Beim TreeWalk kann einem im Prinzip jede im System angelegte Maske über den Weg laufen, dann ist es oft nicht mehr praktikabel, diese alle als zusätzliche Masken einzutragen. In diesem Fall kann man auch direkt mit dem Sord Objekt arbeiten. Die Variable „EM_MASK_LOADED“ enthält die Maskennummer, wenn die Verschlagwortung in die globalen Variablen einer angegeben Maske geladen werden konnte

und den Wert -1, wenn das nicht möglich war, weil eine unbekannte Maskennummer vorliegt. Die Variable EM_INDEX_LOADED ist true, wenn die Maske bekannt war und false, wenn nicht.

Das Auslesen der aktuellen Werte direkt aus dem Sord Objekt ist unproblematisch. Etwas aufmerksamer muss man beim Schreiben sein. Wenn die Maske unbekannt war, kann man direkt in das Sord Objekt schreiben. Falls sie aber vordefiniert war, werden zur Laufzeit die globalen Variablen ins Sord Objekt zurückgeschrieben und Ihre Änderungen würden direkt wieder überschrieben werden. Für solche Masken müssten Sie also die globalen Variablen füllen und nicht das Sord Objekt. Das würde die Programmierung erheblich umständlicher machen – aber es gibt einen Ausweg: Wenn Sie in Ihrer Rule den Wert von EM_INDEX_LOADED auf false umstellen, geht die Laufzeitumgebung davon aus, dass die globalen Variablen nicht gefüllt wurden, und verzichtet auf das Überschreiben im Sord Objekt.

> Die maskenunabhängigen Daten wie NAME, OBJCOLOR, OBJDESC, OBJTYPE und DOCDATE werden immer in die globalen Variablen geschrieben und von dort zurückgeschrieben. Diese Werte sollten Sie also nicht im Sord Objekt, sondern in den NAME- und DOCDATE-Variablen verarbeiten.

Bei Workflow-Rulesets wird die Suche automatisch auf die angegebene Maske beschränkt. Wenn man Workflows zu allen Masken bearbeiten möchte, muss man in der onstart Funktion den Wert EM_ALLOWALLMASKS auf true setzen.

[10.1] Tiff Dokumente klammern

Ab der Version 10.1 setzt der ELOas eine andere Grafiklibrary ein um Tiff-Dokumente zu klammern. Die originale Java Library hat beim Betrieb als Sevlet manchmal Probleme beim Nachladen von PlugIns. Insbesondere im gemeinsamen Betrieb mit den Aspose Libraries gab es hier nicht erklärbare Ausfälle. Weiterhin hat die Java Library die Bilder eingelesen und wieder herausgeschrieben. Dafür mussten sie dekomprimiert und erneut komprimiert werden.

Die neue Library verzichtet auf diesen Aufwand, indem sie direkt auf der Tiff Directory Ebene statt auf Image Ebene arbeitet. Das geht schneller und zuverlässiger. Wenn man die neue Library verwenden will, muss man mergeTiffFiles2 statt mergeTiffFiles aufrufen.

```
ELOAsTiffUtils.mergeTiffFiles2(File[] sourceTiffFiles, File outputFile)
```

JavaScript allgemein

ERSTES ARRAY ELEMENT LÖSCHEN

Für JavaScript Befehle gibt es den Befehl "shift", er löscht das erste Element aus einem Array. Das ist manchmal recht praktisch.

Wenn man den Aufruf auf ein Java Array anwendet, bekommt man keine Fehlermeldung - obwohl es den Befehl in Java nicht gibt. Man bekommt auch keine Behandlung wie in JavaScript. Stattdessen bekommt man einen nutzlosen Mittelweg. Die Arraylänge bleibt gleich, der erste Eintrag wird gelöscht, alle weiteren Einträge werden eins nach vorne gezogen und der letzte Eintrag ist doppelt.

Streng genommen ist das kein Bug in der Rhino Engine sondern ein Programmierfehler im Script. Aber die Rhino Engine könnte so freundlich sein und eine Exception werfen. Da sie das nicht tut, kann ich nur auf das unschöne Verhalten hinweisen. Echte JavaScript Arrays arbeiten problemlos mit dem Shift Befehl.

PROPERTY ODER FUNKTIONSSCHREIBWEISE

In vielen Beispielen hier wird die Property-Schreibweise verwendet, z.B. sord.name und nicht sord.getName(). Das ist eine Komfortfunktion der Rhino Engine. Wenn auf Java Objekte zugegriffen wird, die eine setXyz und getXyz Funktion bieten, dann kann man stattdessen in JavaScript auch direkt das Property xyz lesend und schreibend verwenden.

Prinzipiell sind beide Schreibweisen gleichberechtigt. Wir haben uns für die Property-Schreibweise entschieden, weil sie klarer und übersichtlicher ist.

Beispiel: Ein Eintrag wird weitergeleitet, sein Name soll deshalb das Präfix WG: erhalten

```
sord.name = "WG: " + sord.name;
```

```
sord.setName( "WG: " + sord.getName() );
```

Im ersten Fall sieht man viel schneller und klarer, was hier passiert. Allerdings ist das unsere Präferenz. Bei Ihren eigenen Skripten können Sie persönlich entscheiden, welche Variante Sie verwenden möchten. Wir raten nur dazu, es konsistent zu machen. Entscheiden Sie sich für eine Variante und verwenden Sie diese dann durchgehend.

ZEILENUMBRÜCHE IN REGULÄREN AUSDRÜCKEN

Wenn man im Java Client (oder im ELOas oder Web Client) Texte mit Zeilenumbrüchen analysieren will und der Zeilenumbruch eine wichtige Positionsangabe ist, kann man nicht einfach mit \n arbeiten.

Die Welt der Zeilenumbrüche ist komplex. Manchmal findet man ein Line Feed, manchmal ein Carriage Return und, in der Windows Welt bevorzugt, beides vor. Darauf muss man bei der Formulierung von regulären Ausdrücken Rücksicht nehmen. Die Schreibweise für "A und in der nächsten Zeile B..." wäre "A(?:\r\n|\n|\r)B...". Zum Glück gibt es dafür aber auch eine Abkürzung: "A\RB..." - also \R als Platzhalter für beliebige Zeilenumbrüche.

[10.1] Switch – case mit Java String Objekten

Da man im Forum erkennen kann, dass regelmäßig Programmierer über dieses Problem stolpern: in JavaScript kann man switch Statements über JavaScript Strings ausführen – aber nicht über Java Strings.

```
var benutzer = String(workspace.userName );

switch (benutzer)  {
  case  "Ad":
    indexDialog.setObjKeyValue("LIEFERANT", "Administrator");
    break;

  case  "HM":
    indexDialog.setObjKeyValue("LIEFERANT", "Heinz Mustermann");
    break;

  default:
    indexDialog.setObjKeyValue("LIEFERANT", "Unbekannt default");
}
```

Ein switch über ein Java String führt immer zur default Option, da der Java String ungleich zu jeden anderen JavaScript String ist! Das ist ein der (wenigen) Stellen, an denen man sich genau klar darüber sein muss, ob man ein Java Objekt oder ein JavaScript Objekt vor sich hat.

[10.1] DATEIZUGRIFF AUS SKRIPTEN

JavaScript kennt von Haus aus keine direkten Dateizugriffe. Aufgrund des Sandbox Modells der Browser ist der Zugriff auf lokale Ressourcen prinzipiell erst mal gesperrt. Ab html5 wurde das etwas ausgeweicht, es gibt nun eingeschränkten Zugriff auf einen speziellen lokalen Speicher, aber ein vollwertiger Filesystem Zugriff ist das trotzdem nicht.

ELO Skripte, welche unter der Rhino Engine laufen (Java Client, ELOas, Indexserver), können auf beliebige Java Klassen zugreifen und hierüber den Dateizugriff abwickeln. Dazu muss man aber erst mal wissen, was überhaupt zur Verfügung steht. Leider ist die Java Dateischnittstelle etwas exotisch, deshalb habe ich hier die wichtigsten Klassen und Methoden kurz aufgelistet:

File

Eine Basis für Dateien und Ordner ist das File Objekt. Hierüber kann man abfragen, ob die Datei existiert und auch Ordner anlegen. Zur Verwendung benötig man ein Packages Import für „java.io“.

```
var myDir = new File("d:\\temp\\dirTest");
if (myDir.exists()) {
```

```
  if (myDir.isDirectory()) {
    workspace.setFeedbackMessage("Verzeichnis ist bereits vorhanden");
  } else {
    workspace.setFeedbackMessage("Verzeichnis kann nicht angelegt werden da es bereits
eine Datei mit diesem Namen gibt");
  }
} else {
  myDir.mkdir();
  workspace.setFeedbackMessage("Verzeichnis wurde angelegt");
}
```

Neben der Methode mkdir, welches ein Verzeichnis anlegt, gibt es auch mkdirs – hiermit können Sie nicht nur ein Verzeichnis anlegen, es wird der komplette Pfad erzeugt, falls er noch nicht existiert. Sie müssen also nicht unbedingt Verzeichnisebene für Ebene durchlaufen und prüfen, ob das Unterverzeichnis bereits existiert.

Das File Objekt hat noch eine Reihe weiterer nützlicher Methoden zur Ermittlung der Dateigröße (length()), Schreibschutz (isWriteable(), setWriteable(), Dateidatum (lastModified(), setLastModified()), Umbenennen (renameTo()) und zum Löschen (delete()).

Files

Die Files Klasse stellt ein paar nützliche Funktionen zum Lesen und Schreiben kompletter Dateien zur Verfügung. Die Verwendung erfordert ein Packages Import für „java.nio.file“.

```
var fileContent = "Das ist mein Dateiinhalt\r\nMit zwei Zeilen und Umlauten: äöü.";
var file = new File(myDir, "myFile.txt");

var bytes = new Packages.java.lang.String(fileContent).getBytes("UTF-8");
Files.write(file.toPath(), bytes, StandardOpenOption.CREATE);

var readBack = Files.readAllBytes(file.toPath());
var content = new Packages.java.lang.String(readBack, "UTF-8");

workspace.setFeedbackMessage(content);
```

Wenn Sie ein temporäres Verzeichnis (im Temp-Dir des Benutzers) benötigen, können Sie es über die Methode createTempDirectory erzeugen (analog dazu gibt es auch eine createTempFile Methode):

```
var tempDir = Files.createTempDirectory("scripttest").toFile();
workspace.setFeedbackMessage(tempDir.path);
```

FileStore

Über die Files.getFileStore() Methode können Sie auch ein paar Filesystem-Parameter zu einem Dateipfad abfragen.

```
var file = new File("d:\\temp");
var store = Files.getFileStore(file.toPath());
var msg = "Total: " + store.totalSpace
        + ", usable: " + store.usableSpace
        + ", unallocated: " + store.unallocatedSpace
        + ", name: " + store.name();

workspace.showInfoBox("FileStore", msg);
```

[12.0] DATUMSFORMATIERUNG

Wenn man eine Jahr/ Monatsorientierte Ablagestruktur erzeugen möchte, muss man aus dem ELO ISO Datum eine Ausgabe der Art „März 2019“ erzeugen. Wenn man häufig unterschiedliche Formate benötigt, sollte man sich mit dem Java SimpleDateFormat vertraut machen, was leider nicht so ganz simple ist. Wenn man nur diese einfache Formatierung benötigt, kann man bei JavaScript bleiben.

Die Funktion formatMonthYear erwartet als Parameter ein ELO ISO Datumsformat (YYYYMMDD) und gibt den Monatsnamen und die Jahreszahl zurück.

Die Funktion testFormatter() dient nur zur Veranschaung der Verwendung und zum Test. Die kann man im endgültigen Skript weglassen.

```
function testFormatter() {
  var result = "";
  for (var i = 1; i <= 12; i++) {
    var isoDate = "2019" + ((i < 10) ? "0" + i : i) + "22";
    var fmtDate = formatMonthYear(isoDate);
    result = result + isoDate + " : " + fmtDate + "<br>";
  }

  workspace.showInfoBox("Date Test", result);
}

var translate = [];
translate["01"] = "Januar";
translate["02"] = "Februar";
```

```
translate["03"] = "März";
translate["04"] = "April";
translate["05"] = "Mai";
translate["06"] = "Juni";
translate["07"] = "Juli";
translate["08"] = "August";
translate["09"] = "September";
translate["10"] = "Oktober";
translate["11"] = "November";
translate["12"] = "Dezember";

function formatMonthYear(eloIsoDate) {
  var month = translate[eloIsoDate.substring(4, 6)];
  return eloIsoDate.substring(0, 4) + " " + month;
}
```

Indexserver-Funktionen

ANMELDUNG KONTROLLIEREN

Wenn Sie einen Anwender pauschal von einer Anmeldung abhalten wollen, dann sollten Sie diesen Anwender über die Admin Console einfach sperren.

Wenn Sie Anwender nur Zeitweise von einer Anmeldung abhalten wollen (z.B. zu bestimmten Uhrzeiten oder an bestimmten Tagen) oder nur bestimmte Clients zulassen wollen, dann geht das am besten mit dem Indexserver Event onBeforeLogin.

Das hat gleich auch den Vorteil, dass es für alle (Indexserver) Clients gilt, nicht nur für den Java Client.

```
function onBeforeLogin(ec, userName, options) {
  if (userName == "OnlyWithWebClient" && options.getClientName() !=
LoginScriptOptionsC.CLIENT_NAME_WEBCLIENT) {
    throw "User may login only with the WebClient";
  }
}
```

INDEXZEILENWERTE BEIM WEITERLEITEN FÜLLEN (1)

In einem Workflow kommt es häufig vor, dass im Verlaufe des Prozesses ein Indexzeilenwert geändert werden muss. Wenn es sich um einen berechneten Wert handelt, muss man ein passendes Skript einfügen, welches die Berechnung durchführt. Oft sind es aber statische Werte, z.B. dass bei Erreichen eines Endzustands der Statuswert in „Abgeschlossen“ geändert wird.

So eine Aufgabe kann man über ein generisches Skript abdecken, welches einmal erstellt wird und anschließend in allen Workflows verwendet werden kann. Das Skript wird in das Ende-Event eingetragen und liest aus dem Kommentarfeld des Workflowknotens die einzutragende Information aus und ändert die Verschlagwortung entsprechend.

Die Formatierung des Kommentarfeldes sieht so aus: Jeder Eintrag erhält eine Zeile, es können also mehrere Verschlagwortungsbegriffe auf einmal geändert werden. Jede Zeile besteht aus drei Teilen, jeweils durch ein Pipe Symbol getrennt: Maskenname|Gruppenname|Text.

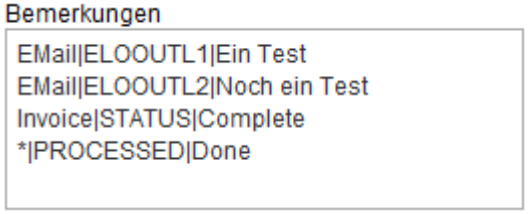


Abbildung 51: Einzutragende Texte im Bemerkungsfeld

Wenn man ein Indexfeld in unterschiedlichen Masken füllen möchte, kann man als Maskenname ein * eingeben. In diesem Fall wird jede Verschlagwortung nach diesem Feld durchsucht. Wenn es gefunden wird, trägt das Skript den neuen Wert ein. Wird es nicht gefunden, ist das kein Fehler – es passiert dann nichts. Das Gleiche gilt, wenn der Maskenname nicht

übereinstimmt. Auch das ist kein Fehler, diese Zeile wird dann übersprungen.

```
function onExitNode( ci, userId, workflow, nodeId ){
  var node = workflow.nodes[nodeId];
  log.debug("This is node: " + node.id + " : " + node.name);
  var desc = node.comment;
  var lines = desc.split("\\R");

  var checkinPending = false;
  var objid = workflow.objId;
  var editInfo = ix.checkoutSord(ci, objid, CONST.EDIT_INFO.mbSord, CONST.LOCK.YES);

  for (var i = 0; i < lines.length; i++) {
    var parts = lines[i].split("\\|");
    if (parts.length != 3) {
      log.info("Invalid entry ignored, number of parts: " + parts.length + " : " + lines[i]);
    } else {
      if (parts[0] == "*") {
        parts[0] = editInfo.sord.maskName;
      }

      if (editInfo.sord.maskName == parts[0]) {
        var keys = editInfo.sord.objKeys;

        for (var j = 0; j < keys.length; j++) {
          var key = keys[j];
          if (key.name == parts[1]) {
            key.data = [ parts[2] ];
            checkinPending = true;
            log.debug("Key found and set: " + parts[2]);
          }
        }
      } else {
        log.debug("Other mask: " + parts[2]);
      }
    }
```

```
  }

  if (checkinPending) {
    ix.checkinSord(ci, editInfo.sord, CONST.SORD.mbLean, CONST.LOCK.YES);
  } else {
    ix.checkinSord(ci, editInfo.sord, CONST.SORD.mbOnlyLock, CONST.LOCK.YES);
  }
}
```

INDEXZEILENWERTE BEIM WEITERLEITEN FÜLLEN (2)

Wenn man einen berechneten Wert in eine Indexzeile füllen möchte, kommt man nicht um ein eigenes Skript herum. Dafür ist es aber auch deutlich einfacher, da es nur eine einzige Aufgabe erfüllt.

```
function onExitNode( ci, userId, workflow, nodeId ){
  var objid = workflow.objId;
  var editInfo = ix.checkoutSord(ci, objid, CONST.EDIT_INFO.mbSord, CONST.LOCK.YES);

  keys = editInfo.sord.objKeys;

  // Hier folgt die Berechnung
  keys[0].data = ["Weitergeleitet am " + new Date()];

  ix.checkinSord(ci, editInfo.sord, CONST.SORD.mbLean, CONST.LOCK.YES);
}
```

KONSISTENZPRÜFUNGEN BEIM SPEICHERN

Es gibt in vielen Prozessen Konsistenzanforderungen, die immer erfüllt sein müssen. Als ein Beispiel: Ein Statusfeld „PRUEFUNG“ darf nur auf „Abgeschlossen“ stehen, wenn das Feld „PRUEFER“ nicht leer ist. Solche Abhängigkeiten kann man im Client per Skript abfragen. Wenn man mit unterschiedlichen Clients arbeitet, kann es aber sehr mühsam werden, sicher zu stellen, dass es wirklich an allen Stellen passiert.

Hier kann es sinnvoll sein, die Prüfung auf den Indexserver zu verlagern. Im Skript-Event onBeforeCheckinSord kann man die Indexzeilen kontrollieren und im Fehlerfall eine Exception werfen. Damit ist sichergestellt, dass die falschen Daten nicht geschrieben werden.

```
function onBeforeCheckinSord(ec, sord, sordDB, parentSord, sordZ, unlockZ) {
  if (sord.mask == 2) {
```

```
    var keys = sord.objKeys;
    if (keys) {
      var hasName = false;
      var isClosed = false;
      for (var i = 0; i < keys.length; i++) {
        var key = keys[i];
        if ((key.name == "ELOOUTL1") && key.data && (key.data.length > 0)) {
          hasName = key.data[0] != "";
        }
        if ((key.name == "ELOOUTL2") && key.data && (key.data.length > 0)) {
          isClosed = key.data[0] == "closed";
        }
      }

      if (isClosed && !hasName) {
        throw new Packages.byps.RemoteException("[ELOIX:" + IXExceptionC.NOT_IX + "]
Cannot close without name");
      }
    }
  }
}
```

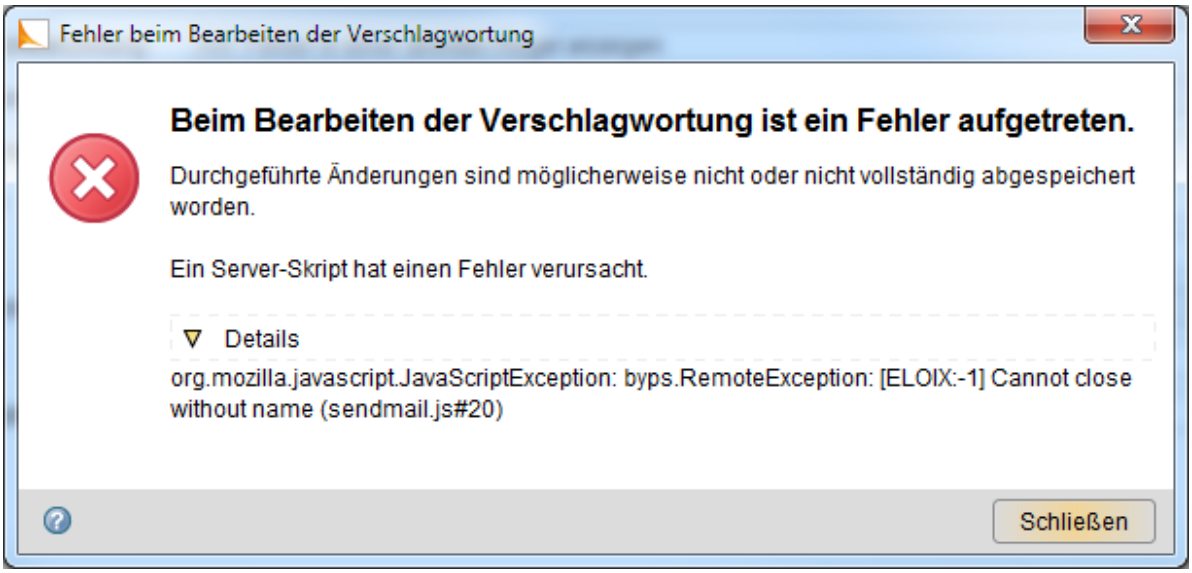


Abbildung 52: Fehlermeldung beim Speichern unzulässiger Daten

Auch bei einer Prüfung auf der Serverseite kann es sinnvoll sein, auf dem Client eine zusätzliche Prüfung einzubauen. Diese hat den Vorteil, dass sie näher am Anwender ist, sie kann z.B. direkt beim Verlassen der Indexzeile erfolgen und eine sprechende Fehlermeldung ausgeben. Die Indexserverprüfung wird erst beim Speichern ausgeführt und liefert eine eher allgemein gehaltene Fehlermeldung.

ANNOTATIONS BEIM CHECKIN LÖSCHEN

Der Anwender hat beim CheckIn die Option vorhandene Annotations zu löschen. Bei manchen Dokumenten möchte man aber sicherstellen, dass beim Einfügen einer neuen Version alle Annotations entfernt werden. Das kann man über ein Indexserver Event machen: onAfterCheckinDocEnd. Das Skriptevent prüft zuerst, ob der spezielle Dokumententyp vorliegt (in diesem Fall anhand des Maskennamens). Anschließend werden die Notizen gelesen und die Annotations ausgefiltert und in ein Array zum Löschen eingetragen. Die Randnotizen bleiben bei diesem Beispiel erhalten.

```
function onAfterCheckinDocEnd(ec, sord, sordDB, parentSord, doc, sordz, lockz) {
  if (sord.maskName == "TestMaske") {
    var id = sord.id;
    var editInfo = ix.checkoutSord(ec.ci, id, CONST.EDIT_INFO.mbNotes, CONST.LOCK.NO);
    var ids = [];
    var notes = editInfo.notes;
    for (var i = 0; i < notes.length; i++) {
      var nt = notes[i].type;
      if ((nt != CONST.NOTE.TYPE_NORMAL) && (nt != CONST.NOTE.TYPE_PERSONAL)
&& (nt != CONST.NOTE.TYPE_STAMP)) {
```

```
        ids.push(notes[i].id);
      }
    }

    if (ids.length > 0) {
      ix.deleteNotes(ec.ci, ids, CONST.LOCK.NO);
    }
  }
}
```

PREVIEWERSTELLUNG FÜR DOKUMENTE

Ob ein Dokument bereits an den Preview-Konverter übergeben wurde, erkennt man in der Tabelle elodmdocs an der Spalte previewsize. Leider gibt es noch kein Indexserver-API zur Steuerung dieses Werts, deshalb muss man hier bei Bedarf manuell in die Datenbank eingreifen.

Wenn ein Dokument noch nicht konvertiert wurde, steht in dieser Spalte der Wert 0. Der Preview-Konverter fragt beim Indexserver regelmäßig diese Liste ab und bearbeitet die Einträge. Wenn dieser nicht konvertiert werden konnte, wird die PreviewSize auf -1 gestellt. Das ist für den Indexserver der Hinweis „es gibt keine Previewdarstellung, aber der Eintrag wurde bereits bearbeitet“.

Wenn ein einzelnes Dokument noch mal erneut konvertiert werden soll, kann man es über dieses Statement auslösen (beachten Sie, dass die docid die Datei-Id und nicht die logische ELO ObjId ist):

```
update elodmdocs set previewsize = 0 where docid = 3450695
```

Wenn eine neue Preview-Konverter Version einen bestimmten Dokumententyp besser behandelt als in der Vergangenheit, kann man auch leicht alle Dokumente eines Typs zur erneuten Konvertierung anmelden:

```
update elodmdocs set previewsize = 0 where ext = 'pdf'
```

Es wäre auch denkbar, dass die neue Version Dokumente erfolgreich konvertieren kann, die von der älteren Version zurückgewiesen wurde. In diesem Fall kann man einfach alle gesperrten Dokumente auf 0 zurücksetzen:

```
update elodmdocs set previewsize = 0 where previewsize = -1
```

[10.1] INDEXSERVERSKRIPTE MIT ERHÖHTEN BERECHTIGUNGEN AUSFÜHREN

Indexserver Eventskripte werden normalerweise mit den Berechtigungen des Anwenders ausgeführt, der das Event auslöst. Somit ist sichergestellt, dass auch im Skript keine Aktionen ausgeführt werden können, auf die der Anwender nicht berechtigt ist.

Manchmal ist aber genau das erwünscht. Für diesen Fall stellt der Indexserver neben der normalen ClientInfo „ci“ auch die Administrative ClientInfo „ciAdmin“ zur Verfügung. Wenn Sie diese ClientInfo im Skript für die Indexserver-Aufrufe verwenden, können Sie mit erweiterten Berechtigungen arbeiten.

```
var subsInfos = ix.checkoutSubs(ciAdmin, userId, LockC.NO);
```

Beachten Sie unbedingt, dass Sie in Ihrem Skript nun selber für die korrekte Behandlung von Berechtigungseinschränkungen verantwortlich sind. Ein **fehlerhaftes** Skript kann einem Anwender vollen Zugriff auf alle Archivobjekte ermöglichen!

[10.2] BLACKLIST FÜR BESTIMMTE DOKUMENTENTYPEN

In einem Security Audit bei einem Kunden wurde bemängelt, dass auch EXE Dateien eingecheckt werden konnten. Wenn ein anderer Anwender diese Datei zur Ansicht öffnet, wird ein ShellExecute(„view“…) aufgerufen. Und „Anzeigen einer EXE“ heißt für Windows „Ausführen“.

Wenn man das verhindern möchte, kann man ein Indexserver Event-Skript hinterlegen, welches die Extension eines neuen Dokuments prüft und im Fehlerfall eine Exception wirft. Aus Usability Sicht nicht die schönste Lösung, da der Anwender mit einer sehr technischen Fehlermeldung konfrontiert wird – aber es erfüllt den Zweck.

```
var forbiddenExtensions = ["exe", "bat", "scr", "xyz"];

function onBeforeCheckinDocEnd(ec, sord, sordDB, parentSord, doc, sordZ, lockZ) {
  log.warn("onBeforeCheckinDocEnd: " + sord.name);
  if (doc.atts) {
    for (var i = 0; i < doc.atts.length; i++) {
      check(doc.atts[i]);
    }
  }

  if (doc.docs) {
    for (var i = 0; i < doc.docs.length; i++) {
      check(doc.docs[i]);
    }
  }
}

function check(doc) {
  var ext = doc.ext.toLowerCase();
```

```
  log.warn("check: " + ext);
  for (var i = 0; i < forbiddenExtensions.length; i++) {
    if (forbiddenExtensions[i] == ext) {
      throw "Invalid document extension";
    }
  }
}
```



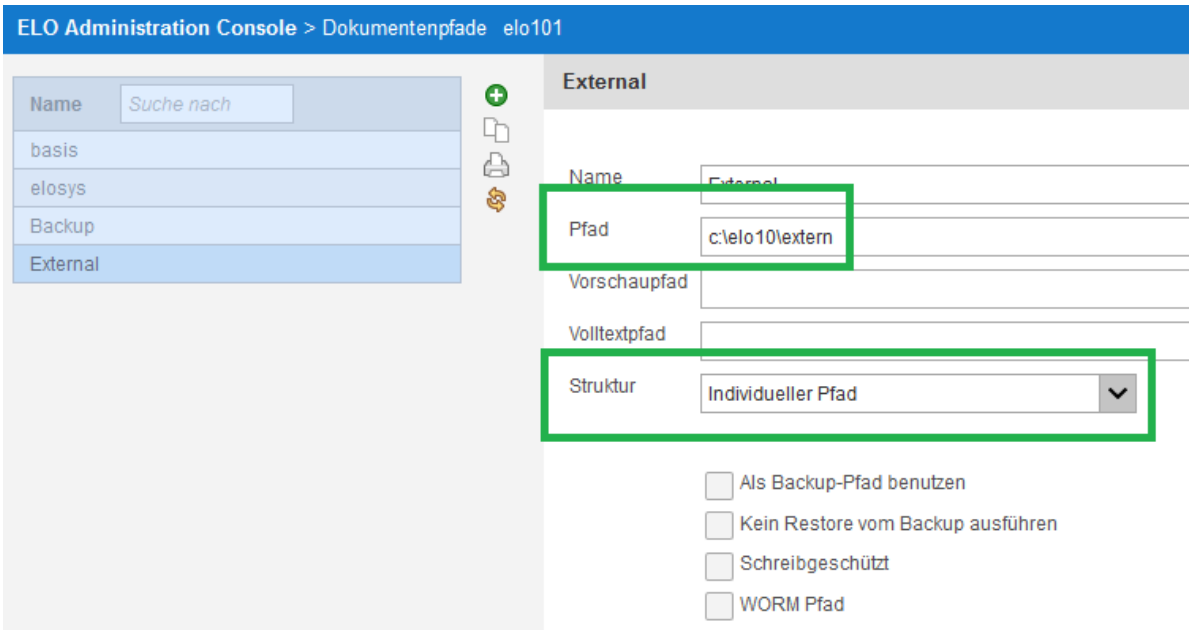
[10.2] EXTERNE DATEIEN IM ARCHIV ANZEIGEN

Wenn man aus einem anderen Archivsystem nach ELO migriert, ist es manchmal nicht erwünscht, dass die Dokumente kopiert werden und somit zumindest temporär doppelt vorhanden sind. Speziell bei WORM Medien oder Hard Disk Appliances mit Retention Period kann das dauerhaft massiv Speicher verschwenden.

Aus diesem Grund gibt es im Indexserver eine Funktion, die es erlaubt, neue Dokumente abzulegen und den Dateipfad statt einer kompletten Datei zu übergeben.

> Beachten Sie, dass ELO keinerlei Kontrolle über die Dateien, den Lebenszyklus dieser Dateien oder den Veränderungsschutz hat. Das ist in diesem Falle alles Aufgabe des Administrators.

Als Vorbereitung muss man den Zugriff auf diese Dokumente so konfigurieren, dass sie vom ELOdm aus sichtbar sind – z.B. über ein Laufwerk oder ein File Share. Anschließend wird in der Admin Console ein Pfad auf diesen Pfad angelegt. Dieser Pfad muss die Strukturinformation „Individueller Pfad“ besitzen. Als Dateipfadinformation geben Sie den Teil des Pfades an, den alle Dokumente gemeinsam besitzen. Im Beispiel C:\elo10\extern.



Im Folgenden möchte ich zeigen, wie die Indexserverfunktion zum Anlegen eines derartigen Dokuments verwendet wird. Das Beispiel läuft im Java Client, ist aber leicht auf den ELOas zu übertragen.

```
function addFile(parentId, name, relpath) {
  var ed = ixc.createDoc(parentId, "0", null, EditInfoC.mbSordDoc);
  var sord = ed.sord;
  sord.name = name;

  var dv = new DocVersion();
  dv.pathId = 4;
  dv.relativeFilePath = relpath;

  var doc = ed.document;
  doc.docs = [dv];

  ixc.checkinDocEnd(sord, SordC.mbAll, doc, LockC.NO);
}

function externalTest() {
  addFile("211081", "Picture 1", "group1\\picture1.png")
  addFile("211081", "Picture 2", "group1\\picture2.png")
  addFile("211081", "Picture 3", "group1\\picture3.png")
```

```
  addFile("211081", "Text 1", "group2\\text1.txt")
}
```

Zuerst wird mittels createDoc ein leeres logisches Dokument erzeugt. Die Verschlagwortung wird hier der Einfachheit halber nur im Namen gefüllt. Anschließend wird ein DocVersion Objekt erzeugt und die Pfad-Id aus dem neu angelegten Pfad eingetragen. Die Admin Console zeigt diese an, wenn man sich die Pfaddefinition anzeigen lässt. Weiterhin wird der Dateipfadanteil in relativePath eingetragen, der nicht von der Pfaddefinition abgedeckt wird. Eine Datei „c:\elo10\extern\group1\picture1.png“ wird also mit dem relativen Pfad group1\picture1.png eingetragen, da der Anfang c:\elo10\extern durch die Pfaddefinition abgedeckt ist. Zuletzt wird das DocVersion Objekt in das Verschlagwortungsobjekt eingetragen und mittels checkinDocEnd gespeichert. Die Dokumentendatei wird in diesem Fall nicht kopiert, sondern der ELOdm legt sich eine Referenz auf die Originaldatei an.

In der Version 10.2 können diese Dokumente nicht in den Volltext abgelegt werden, da der ELOdm die Volltextdatei nicht speichern kann. Das wird sich in der Version 11 ändern.

Formulare

NACHRICHT AN DEN CLIENT SENDEN

Ein Browserskript wird in einer Sandbox ausgeführt und kann deshalb kaum lokale Aktionen durchführen. Um die Integration in die Umgebung und den Client zu verbessern, besteht die Möglichkeit, dass ein Formularskript Nachrichten an ein Client-Skript versendet. Die benötigten lokalen Aktionen können dann vom Client-Skript ausgeführt werden.

Das Beispiel verwendet ein kleines Formular mit einem Eingabefeld MSG_TEXT für die Nachricht und einem Button JS_SEND zum Triggern der Aktion. Die Antwort des Clientskripts wird in einem weiteren Feld dargestellt (MSG_RESPONSE).

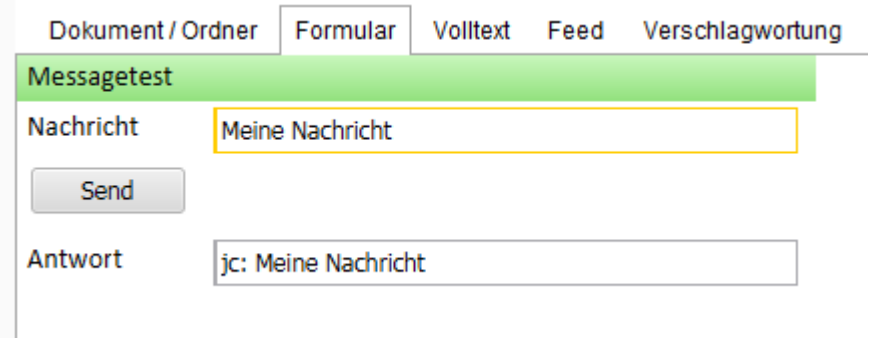


Abbildung 53: Formular mit Eingabefeld für Client-Skript

Zum Testen wird das Nachrichtenfeld gefüllt und der Button Send gedrückt. Daraufhin wird das Formularskript gestartet, welches das Textfeld ausliest und eine Nachricht erzeugt. Diese wird dann an das Client Skript gesendet welches wiederum eine Antwort erzeugt, die in dem MSG_RESPONSE Feld angezeigt wird. Im Beispiel werden nur einfache Strings übertragen, prinzipiell können aber beliebige Objekte versendet werden.

```
function JS_SEND() {
  var message = $val('MSG_TEXT');
  var data = {text: message};
  api.communication.Parent.sendCustomMessage('ThmMsg', data, function(data, event){
     $update('MSG_RESPONSE', data.response);
  });
};
```

Das eigentliche Versenden erfolgt über die Methode sendCustomMessage. Der erste Parameter enthält einen Namen für die Nachricht, da unterschiedliche Quellen Daten schicken können und der Empfänger die für ihn wichtigen erkennen können muss. Der zweite Parameter enthält das zu übertragende Datenobjekt.

Der dritte Parameter ist für einen klassischen Skriptentwickler etwas merkwürdig, für einen Web Entwickler normal – es ist eine Callback Funktion, die aufgerufen wird, wenn das Client-Skript die Antwort sendet. Diese Vorgehensweise ist im Web Bereich üblich. Jede Aktion, die etwas länger dauern kann, wird nur abgeschickt aber nicht auf die Antwort gewartet. Das Skript läuft sofort weiter. Für die

Antwort wird eine Callback Funktion hinterlegt, die dann erst aktiv wird, wenn die Antwort da ist. Man vermeidet so, dass eine langlaufende externe Aktion das komplette Browserscripting blockiert.

Die Callback Funktion hat zwei Parameter – der erste enthält das Datenobjekt, welches das Client-Skript zurückgesendet hat. Der Zweite enthält ein event Objekt.

Im Client wird eine Eventfunktion hinterlegt, die aufgerufen wird, wenn vom Browser eine Nachricht kommt. Diese prüft zuerst, ob der Name der Nachricht für dieses Skript gedacht ist, nur dann wird es aktiv.

```
function eloReceiveBrowserMessage(msg, compName){
  if (msg.name == "ThmMsg") {
    var respObject = {response: 'jc: ' + msg.data.text};

        var browserComp = components.getClientBrowserComponent(compName);
        browserComp.sendCustomResponse(msg, respObject);
  }
}
```

Als Antwort wird ein Objekt mit einem Property response angelegt. Der Wert wird auf den festen String „jc:“ Und dem gesendeten Text gesetzt. Somit arbeitet das Client Skript als Echo Funktion. Anschließend ermittelt das Client-Skript die Quelle und ruft dort die Methode sendCustomResponse auf, um die Antwort zu versenden.

NACHRICHT AN EIN FORMULAR SENDEN

Das Senden einer Nachricht aus einem Client-Skript an ein Formular funktioniert ähnlich. Im Beispiel habe ich das Formular um ein Ausgabefeld MSG_CALL erweitert, welches die Clientnachricht anzeigt.

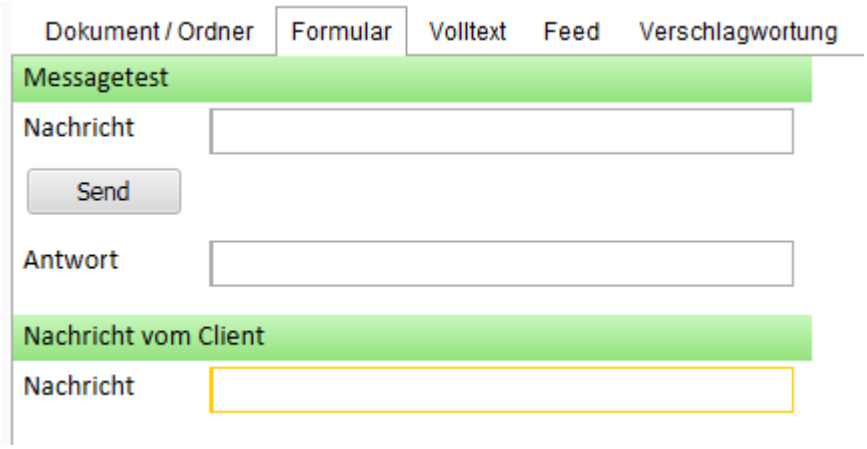


Abbildung 54: Formular mit Ausgabefeld für Client-Skript

Im Client gibt es ein Button-Klick Event, welches dann eine Nachricht an das aktuell sichtbare Formular sendet.

```
var count = 2;
function eloScriptButton100Start(){
  var browserComp = components.getClientBrowserComponent(
            CONSTANTS.BROWSER_COMPONENT_NAME.FORMULAR_PANEL);

  var fctParamsObject = {
        text: 'Die ist ein Text aus dem Java Client',
        count: count++
  };

  browserComp.callCustomFunction( "ThmCall", fctParamsObject, "browserResponse" );
}
```

Das Skript ermittelt zuerst das aktuelle Browser-Fenster. Anschließend wird das Nachrichten-Objekt erzeugt und mit den Werten „Das ist ein Text aus dem Java Client“ und einem fortlaufenden Zähler gefüllt. Zuletzt wird das Objekt unter dem Namen „ThmCall“ an den Browser geschickt.

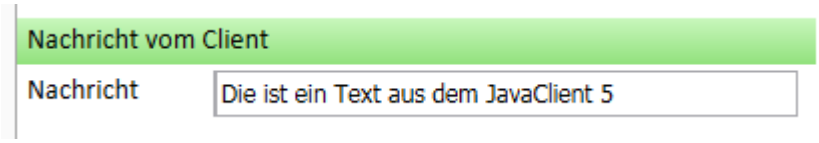


Abbildung 55: Anzeige im Formular

Für die Antwort des Browsers wird die Callback Funktion „browserResponse“ angemeldet.

```
function browserResponse(data, msg) {
      workspace.showInfoBox("Response", data.response);
}
```

Die Callback Funktion zeigt einfach nur den zurückgegeben Wert in einer InfoBox an.

Im Browser muss eine Event-Funktion angemeldet werden, die für Nachrichten mit dem Namen „ThmCall“ aufgerufen wird. Das geschieht durch die Methode onCustomMessage. Sie bekommt zwei Parameter – den zu überwachenden Messagen-Namen und eine Callbackfunktion.

```
api.communication.Parent.onCustomMessage('ThmCall', function(data, event){
      $update('MSG_CALL', data.text + ' ' + data.count);
      api.communication.Parent.sendResponse(
          event, {response: data.text + '  ' + new Date()}
```

```
    );
});
```

In der Callbackfunktion wird zuerst der übergebene Text in das Ausgabefeld MSG_CALL des Formulars übertragen. Anschließend wird die Antwort zurückgesendet, das macht der Aufruf sendResponse. Im Beispiel wird an den übergebenen Text einfach ein aktuelles Datum angefügt.

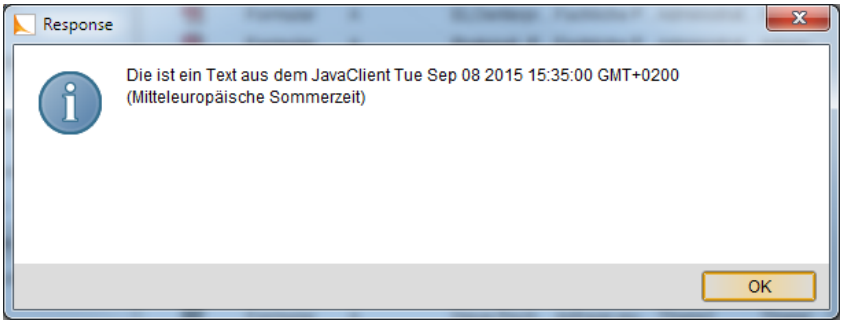


Abbildung 56: Rückgabewert anzeigen

SPALTENBREITE IN EINEM FORMULAR SETZEN

Ein Formular besteht aus einem 2-dimensionalen Grid mit Ein- und Ausgabeelementen. Im Formulareditor können Sie zwar die Größe der Elemente beeinflussen (z.B. über die Anzahl der Zeichen in einem Eingabefeld). Sie können aber nicht direkt die Breite einer Spalte bestimmen, diese wird normalerweise dynamisch aus dem Inhalt bestimmt.

Das ist über einen CSS Eintrag unter „Benutzerskripte bearbeiten“ möglich. Da das Grid aus einem TABLE Element besteht, können Sie hierfür einen CSS Eintrag erstellen, der die Breite des n-ten TD Eintrags festlegt.

Im Beispiel wird die zweite Spalte auf eine Breite von 600 Pixel voreingestellt. Beachten Sie bitte, dass der Browser bei Platzmangel durchaus von diesem Wunsch abweichen kann.

```
<style type='text/css'>

td:nth-child(2) {
        width: 600px;
}

</style>
```

UMFANGREICHE SKRIPTE IN FORMULAREN

Der Formulardesigner bietet die Möglichkeit, über die Funktion “Benutzerskripte bearbeiten”, eigene Skriptfunktionen in ein Formular einzubinden. Das ist für kleinere Skriptaufgaben einfach und schnell

erledigt. Bei umfangreichen Skripten möchte man aber stärker Strukturieren und einen richtigen JavaScript Editor verwenden. In diesem Fall ist es sinnvoll, das umfangreiche Skript in eine (oder mehrere) eigene Datei auszulagern.

Die Skriptdatei können Sie im Ordner „ELOwf Base/Webapp“ abspeichern. Im Frame Dokument müssen Sie jetzt noch den Skriptdateinamen eintragen. Danach können Sie die Skriptfunktionen direkt verwenden.

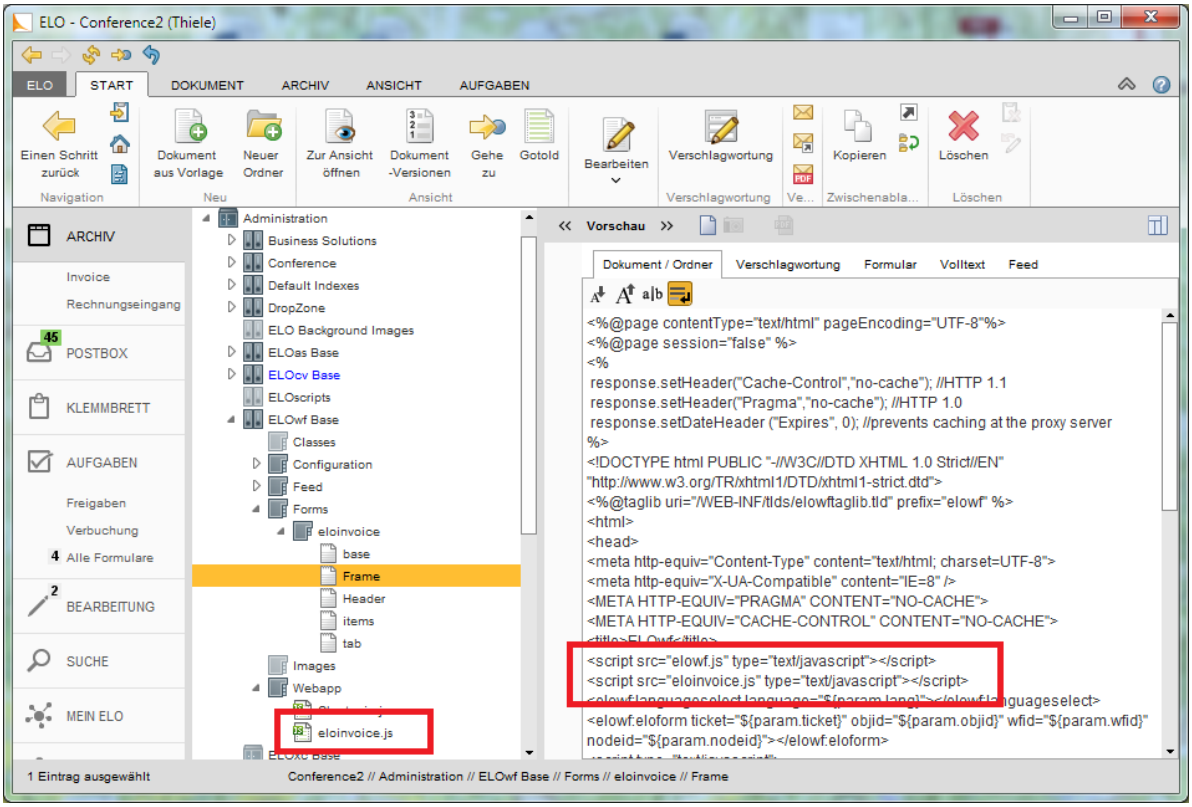


Abbildung 57: Eigene Skriptdateien in ELOwf Formularen

Beachten Sie bitte, dass die Browser solche Skriptdateien in einem Cache halten. Das kann besonders in der Debugging-Phase schwierig sein. Wenn Änderungen im Skript scheinbar nicht aktiv werden, sollten Sie den Browser-Cache löschen.

CHECKBOX ALS STEMPEL VERWENDEN

Diese Skriptfunktion zeigt, wie man eine Checkbox so ansteuern kann, dass sie zwar gesetzt, aber nicht wieder zurückgesetzt werden kann. Ein Beispiel für so eine Funktion wäre „Rechnungsprüfung erfolgt“. Der Anwender bestätigt es durch Klicken der Checkbox, kann es aber später nicht einfach wieder zurücknehmen. Zudem wird in einem Textfeld der Name des Bearbeiters zusammen mit dem Datum gespeichert.

Die Verwendung ist einfach. Im Skriptevent „inputChanged“ wird für jede überwachte Checkbox diese Funktion aufgerufen. Als Parameter wird der source Parameter des Events weitergereicht sowie die Namen des Checkbox- und des Bearbeiterfelds übergeben.

```
lockedCheckbox(source, "IX_MAP_CHECKED", "IX_MAP_CHECKEDBY");
```

Die Funktion prüft beim Programmstart (source == null), ob die Checkbox aktiv ist. In diesem Fall wird sie auf ReadOnly gestellt, damit sie nicht mehr verändert werden kann. Wenn die Checkbox während der Bearbeitung gesetzt wird, trägt die Funktion den Namen und die Uhrzeit in das Bearbeiterfeld ein und sperrt die Checkbox. Das Bearbeiterfeld sollte bereits im Formulardesigner als ReadOnly eingestellt werden. Es muss niemals manuell bearbeitet werden.

```
// Verwaltet eine Checkbox die nur aktiviert aber nicht
// mehr deaktiviert werden kann. Zur Quittierung von
// Zuständen, die nicht mehr zurückgenommen werden kann.
//
// source : Input Element der Checkbox
// cbName : Name der Variablen mit dem Check-Zustand
// confirmName : Name der Variablen mit dem Signaturtext
// enabled : Checkbox freigeben
//
function lockedCheckbox(source, cbName, confirmName, enabled) {
  if (source) {
    // Bearbeitung
    if (source.name == cbName) {
      if (source.checked) {
        $update(confirmName, ELO_PARAMS.ELO_CONNECTUSERNAME + " - " + toDay());
      }
    }
  } else {
    // Initialisierung
    var check = $var(cbName);

    if (check.checked && ($val(confirmName) == "")) {
      check.checked = false;
    }
```

```
    if (check.checked || !enabled) {
      check.disabled = true;
    }
  }
}
```

NAME AUS ARRAY-VARIABLEN ERMITTELN

Arrays über Indexzeilen oder Map Felder werden im ELOwf mittels einer durchlaufenden Nummerierung im Namen erzeugt: Kunde1, Kunde2 ...

Wenn man einen Namen und einen Index hat, ist es recht leicht, den Map-Feldnamen zu erzeugen:

var mapName = name + index;

Der umgekehrte Weg ist etwas aufwändiger. Wenn man den kombinierten Namen hat – wie lautet dann der Basisname? Die folgende Funktion sucht in einem String die letzte Ziffernfolge und trennt diese ab. Es bleibt dann der Basisname über.

```
// Liefert den Namensanteil ohne Zeilennummer zurück.
getNameFromName : function(name) {
  var pos = name.search(/\d+$/);
  if (pos > 0) {
    name = name.substring(0, pos);
  }

  return name;
}
```

INDEX AUS ARRAY-VARIABLEN ERMITTELN

Analog zur Abtrennung des eigentlichen Namensteils aus einer Arrayvariablen kann man auch den Index abtrennen. Die Vorgehensweise ist wie bei der Namensabtrennung, nur wird hier der Ziffernanteil abgetrennt und in die parseInt Methode übergeben damit man als Rückgabe eine Number und kein String erhält.

```
function getIndexFromName( name ) {
  var pos = name.search(/\d+$/);
  if (pos > 0) {
    name = name.substring(pos);
    return parseInt(name, 10);
```

```
  }


  return 0;
}
```

ZUGRIFF AUF DEN INDEXSERVER AUS EINEM FORMULAR

Ein Formular besitzt in den ELO_PARAMS den Zugriff auf das Ticket des aufrufenden Clients. Dieses Ticket kann dafür verwendet werden, dass sich das Formular eine JSON Indexserververbindung öffnet – ohne dass der Anwender erneut ein Passwort eingeben muss.

Das Beispiel öffnet beim Start des Formulars eine Indexserver-Verbindung und holt die Versionsnummer des Indexservers ab. Diese wird in dem Textfeld mit dem Namen MSG_VERSION ausgegeben. Die Indexserverbefehle und Objekte können nur verwendet werden, wenn man die Datei EloixClient.js in die Seite einbindet.

```
<script type="text/javascript" src="/ix-elo90-web/EloixClient.js"></script>
…

 function inputChanged(source) {
   if (!source) {
     var fact = new de.elo.ix.client.IXConnFactory('/ix-elo90-web/ix', 'WFScript', '1.0');
     var con = fact.createFromTicket(ELO_PARAMS.ELO_TICKET);
     var version = con.getVersion();
     $update('MSG_VERSION', version );
   }
 }
```

Die JSON Verbindung verwendet die Anmeldung des Clients. Deshalb wird sie automatisch geschlossen, wenn der Client beendet wird. Aus diesem Grund kann man so eine URL nicht an andere Anwender weitergeben.

FILTER FÜR FELDINHALTE

Sie können zu Formularfeldern eine vorgegebene Standardvalidierung eingeben, um bestimmte Eigenschaften zu prüfen. Es gibt aber auch die Möglichkeit, dass Sie hier eine Filterfunktion eintragen um den Feldinhalt direkt bei der Eingabe zu prüfen und zu verändern.

Hierzu erzeugen Sie zuerst die JavaScript Funktion – am einfachsten im Formularskript – und tragen den Namen der Funktion dann im Validierungs-Feld des Eingabefelds ein. Hierüber kann man Zeichen von der Eingabe ausschließen oder Zeichen direkt bei der Eingabe umwandeln.

```
function JS_FILTER_Uppercase(front, inserted, back) {
  return inserted.toUpperCase();
}
```

Diese Funktion wird bei jedem Tastendruck aufgerufen und bekommt als Parameter „inserted“ den neu hinzugefügten Feldinhalt. Die Parameter front und back enthalten den Text vor bzw. hinter dem neuen Text (falls das neue Zeichen in der Mitte eingefügt wird). Der Rückgabewert bestimmt dann den tatsächlichen neuen Feldinhalt.

Eine weitere Verwendungsmöglichkeit wäre z.B. eine Seriennummerneingabe, welche nur Ziffern erlaubt. Buchstaben werden automatisch ausgefiltert, ein O (Großbuchstabe O) wird in eine 0 (Null) umgewandelt und ein l (kleines L) wird in eine 1 (Eins) umgewandelt.

```
function JS_FILTER_Uppercase(front, inserted, back) {
  if ((inserted >= '0') && (inserted <= '9')) {
    return inserted;
  }

  if (inserted == 'O') {
    return '0';
  }

  if (inserted == 'l') {
    return '1';
  }

  return '';
```

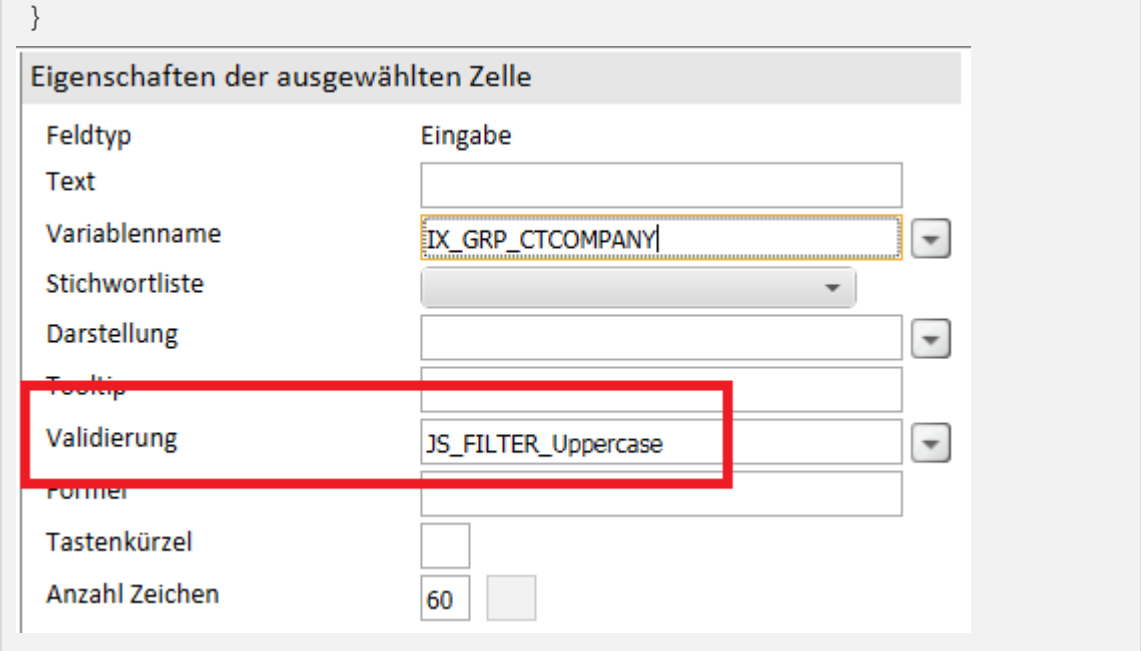


Abbildung 58: Eintragen der Filterfunktion

Wenn Sie zu Kontrollzwecken den kompletten Feldinhalt benötigen und nicht nur den neuen Teil, können Sie ihn durch var all = front + inserted + back zusammensetzen.

TAB SPERREN

Man kann einzelne Tabs im Formular in Abhängigkeit des aktuellen Status oder Anwender sperren. Dafür gibt es eine Hilfsfunktion setTabStatus. Als Parameter gibt man den Namen des Tab-Templates und den Namen der zu sperrenden Tab Seite (ist auch ein Template) an.

```
function inputChanged(source) {
  var ena = $val("IX_NAME") == "test";
  setTabStatus("repos", "more", ena);
}
```

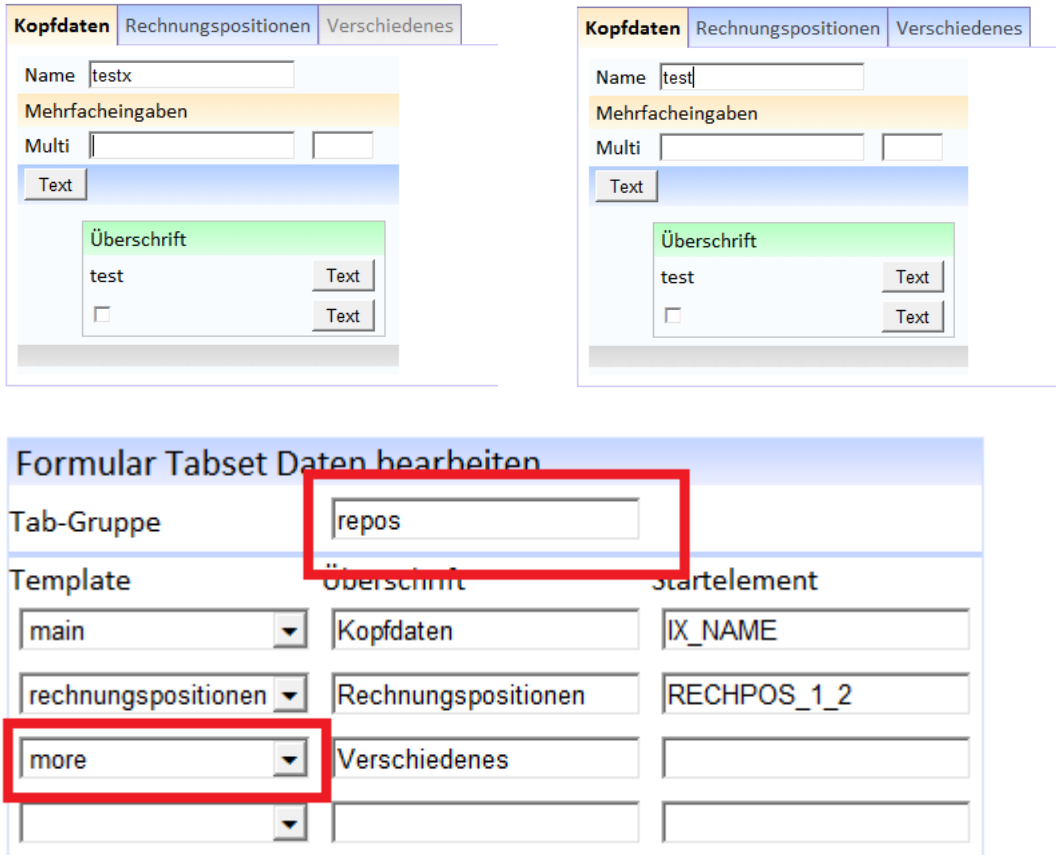


Abbildung 59: Tab sperren

[10] EIGENE BILDER IM FORMULAR ANZEIGEN

Wenn man Bilder aus dem ELO Archiv im Formular anzeigen will, geht das recht einfach über das Bild Element im Formulardesigner. Damit kann man aber keine Bilder aus einem Web Server anzeigen und noch weniger unterschiedliche Bilder in Abhängigkeit von der aktuellen Verschlagwortung.

So eine Funktion lässt sich recht einfach per Skript realisieren. Für die Berechnung der URL aus der Verschlagwortung würde man ohnehin ein Skript benötigen, so dass der Zusatzaufwand minimal ist.

Im Formular legt man wie gewohnt ein Bildelement an der passenden Stelle an. Das Bildelement muss einen Namen erhalten, auch dann, wenn es nicht eindeutig einer Indexzeile zugeordnet werden kann. Der Name wird benötigt, damit man später im Skript darauf zugreifen kann. Mein Bildelement im Beispiel nenne ich XYZ.

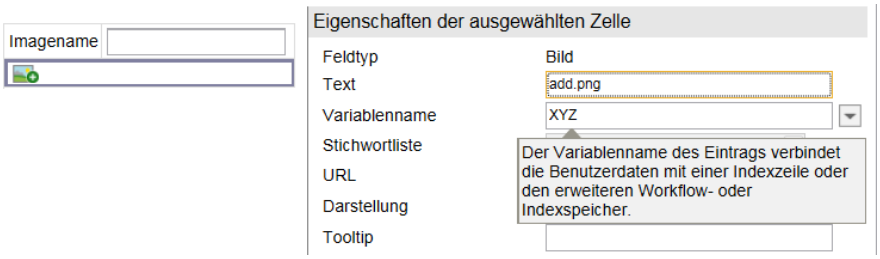


Abbildung 60: Benanntes Bildelement im Formular

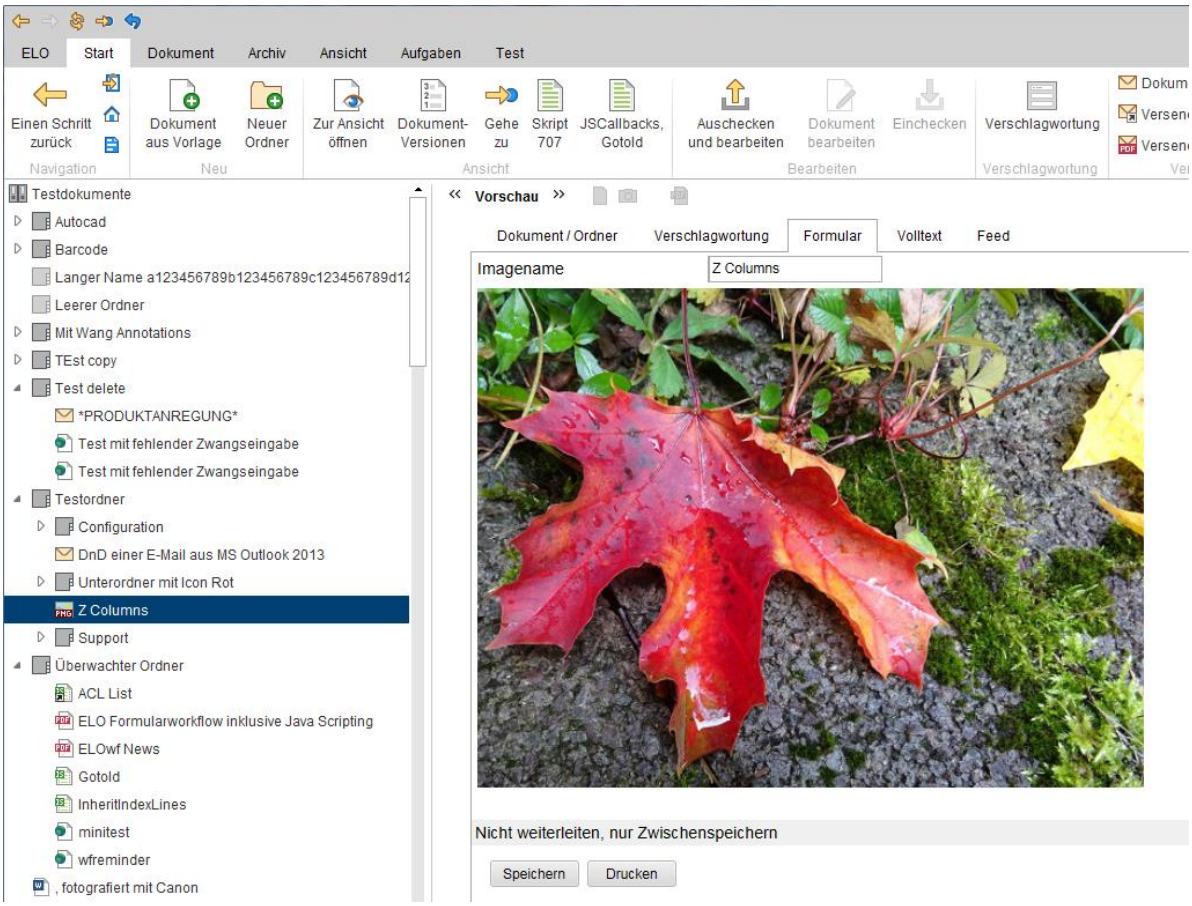
Als Nächstes geht man in den Skripteditor zum Formular. Dort gibt es ein Skriptereignis inputChanged. Beim Start der Anzeige wird es zum ersten Mal aufgerufen, der Parameter source ist zu diesem Zeitpunkt (und nur zu diesem Zeitpunkt) null. Dort kann man eine Funktion hinterlegen, welche die Indexinformation auswertet und daraus eine URL berechnet. In dem einfachen Beispiel wird das erste Zeichen der Kurzbezeichnung verwendet und je nachdem, ob diese mit einer Ziffer, den Buchstaben A...H oder einem andern Zeichen beginnt, jeweils ein anderes Image angezeigt.

```
function inputChanged(source) {
  if (!source) {
    adjustImage()
  }
}

function adjustImage() {
  var images = document.getElementsByName("XYZ");
  if (images.length > 0) {
    var select = ELO_PARAMS.IX_NAME.charAt(0).toUpperCase();
    var url = "http://forum.elo.com/images/Leaf.jpg";

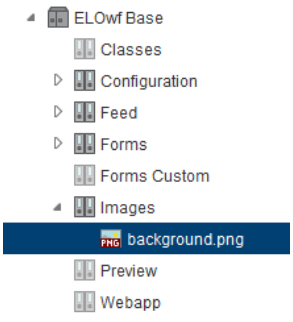
    if ((select >= '0') && (select <= '9')) {
      url = "http://forum.elo.com/images/BurgCochem.jpg"
    } else if  ((select >= 'A') && (select <= 'H')) {
      url = "http://forum.elo.com/images/Hagnau.jpg";
    }

    images[0].src=url;
  }
}
```



[10.1] EIGENE HINTERGRUNDBILDER IM FORMULAR

Es ist relativ einfach möglich, im Formular eigene Hintergrundbilder einzublenden. Hierzu muss man einerseits das Bild selber zur Verfügung stellen. Das geht über dem images Ordner im ELOwf. Beachten Sie bitte, dass der Name in der Kurzbezeichnung die Extension beinhalten muss, also z.B. „background.png“. Dokumente, die man hier ablegt, werden im WebServer im Ordner /images zur Verfügung gestellt.



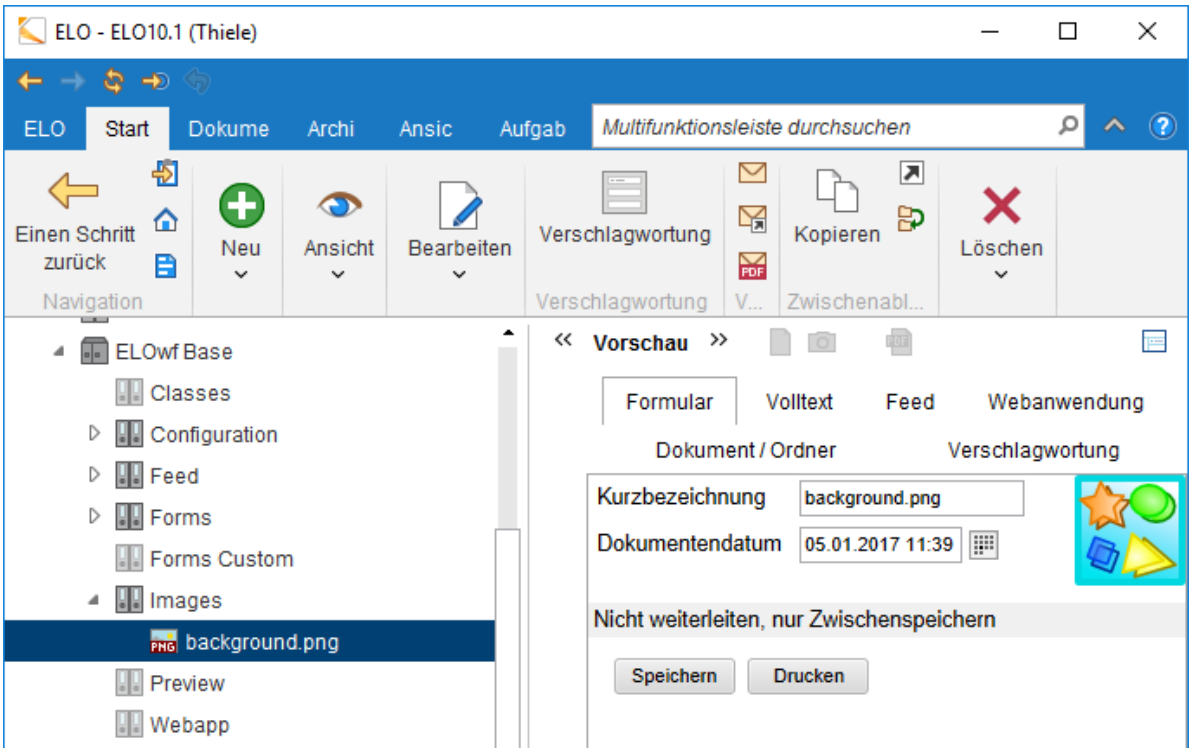
Weiterhin muss man das Bild per CSS im body Tag als background-image anmelden. Das kann man entweder global im Frame Dokument für alle Formulare machen. Oder im Formular selber unter „Benutzerskripte bearbeiten“.

Dort gibt einen vorkonfigurierten Bereich „<style>“ – den füllt man mit den Namen des Bildes und der Formatierung aus:

```
<style type='text/css'>
  body {
```

```
    background-image: url("images/background.png");
    background-repeat: no-repeat;
    background-position: right top;
  }
</style>
```

Nach einem refresh des ELOwf steht das Hintergrundbild dann im Formular zur Verfügung. Da es ein Hintergrundbild ist, wird es von allen Formularelementen überlagert. Wollen Sie das Bild im Vordergrund haben, können Sie ein normales Image Formularelement verwenden.



[11.0] FORMULARFELD IN ABHÄNGIGKEIT EINES WERTES EINFÄRBEN

In der Community wurde die Frage gestellt, ob man ein Formularfeld in Abhängigkeit eines Wertes aus einem anderen Feld einfärben könnte. Das kann man mit einem kleinen Skript erledigen.

Die Funktion updateColor prüft zuerst, ob das Werte Feld geändert wurde oder ob das Formular initial geladen wird (source == null). In diesen Fällen muss der CSS Klassenname in das Zielfeld passend eingetragen werden. Wurde ein anderes Feld verändert, muss die Funktion nicht aktiv werden und wird sofort verlassen.

Falls das Zielfeld weitere CSS Klassen besitzt, kann man den Namen nicht fest eintragen, sondern man muss ihn ergänzen bzw. ersetzen. Das Beispiel geht davon aus, dass keine weiteren Klassen vorhanden sind.

```
function inputChanged(source) {
  updateColor(source);
}

function updateColor(source) {
  if (!source) {
    source = $var("IX_GRP_ELOOUTL1");
  } else if (source != $var("IX_GRP_ELOOUTL1")) {
    return;
  }

  var text = source.value;
  var nameField = $var("IX_NAME");

  if (text.indexOf("Thiele") >= 0) {
    nameField.className = "formred";
  } else {
    nameField.className = "formgreen";
  }
}
```

Indexserver Events

WORKFLOWWEITERLEITUNG ERZEUGT RANDNOTIZ

Das Kommentarfeld beim Workflow-Weiterleiten war ursprünglich als Feld für eine Arbeitsanweisung gedacht. Es ist ausschließlich auf diesen Knoten bezogen und weder vom Vorgänger beschreibbar, noch vom Nachfolger einsehbar. Unglücklicherweise ist das für den Anwender nicht offensichtlich. Deshalb tragen manche Benutzer dort Texte als Nachricht für den Nachfolger ein.

Im Prinzip ist der richtige Ort für solche Informationen der Feed. Er wird aber nicht von allen Kunden aktiv verwendet. Deshalb wurde dieses Skript entwickelt:

Wenn das Skript in das Ende-Event eines Personenknotens eingetragen wird, liest es das Kommentarfeld des Knotens aus und – wenn es nicht leer ist – erzeugt eine Randnotiz mit diesem Text. Diese Randnotiz ist dann für den nächsten Anwender sofort sichtbar.

```
function onExitNode(ci, userId, workflow, nodeId) {
  var objid = workflow.objId;
  var node = workflow.nodes[nodeId];
  var desc = node.comment;
  if (desc) {
    var note = ix.createNote2(ci, objid, CONST.NOTE.TYPE_NORMAL, "");
    note.desc = desc;

    ix.checkinNotes(ci, objid, [note], CONST.NOTE.mbAll, CONST.LOCK.NO);
  }
}
```

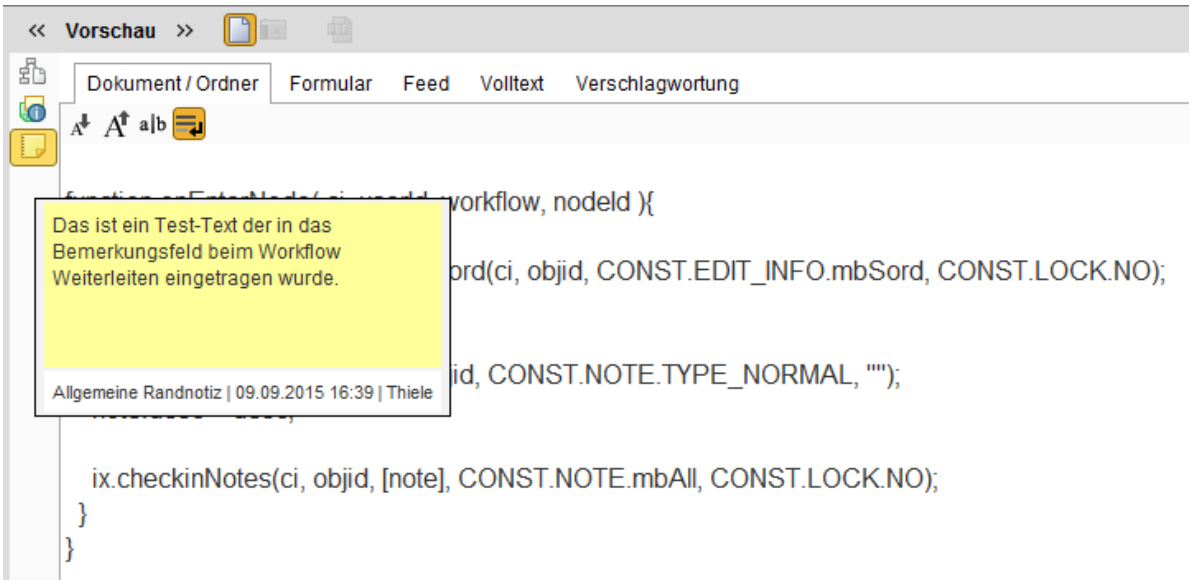


Abbildung 61: Durch Eventskript erzeugte Randnotiz

MAIL FÜR NEUE WORKFLOWTERMINE

Wenn ein Anwender nicht regelmäßig im ELO arbeitet, sieht er auch nicht, wenn er einen neuen Workflowtermin bekommen hat. Wenn diese Benachrichtigung aber zeitnah gebraucht wird, ist es sinnvoll, ihm per Mail eine Nachricht zu senden, dass eine neue Aufgabe vorliegt.

Hierfür wird das Indexserver-Event onEnterNode herangezogen. Es wird aktiv, wenn ein Anwender eine neue Aufgabe zugeteilt bekommt. In dem Skript wird zuerst geprüft, ob ein Anwender nicht an sich selber weiterleitet (dann wird keine Mail verschickt). Anschließend wird eine Mail mit einer elodms Referenz erzeugt und die Zieladresse aus den ELO Anwenderdaten ermittelt.

```
unction onEnterNode( ci, userId, workflow, nodeId ){
  try {
    var node = getNodeById(workflow, nodeId);
    if (node && (userId != node.userId)) {
      var mail = getUserAddress(ci, node.userId);
      var subject = "Workflowaufgabe: " + node.name;
      var body = '<h1>' + node.name + '</h1><h4>' + workflow.name + '</h4><a href =
"elodms://wf/' + workflow.id + '/' + node.id + '">Sie haben eine neue Workflowaufgabe
erhalten.</a>';
      sendMail(mail, subject, body);
    }
  } catch(ex) {
    log.info("Error processing Workflow Info: " + ex);
```

```
  }
}

function sendMail(to, subject, body) {
  var sendMail = new Packages.de.elo.ix.jscript.SendMail("mail/SRVPEMAIL02vm");
  sendMail.subject = subject;
  sendMail.sender = "eloservice@elo.com";
  sendMail.TO = to;
  sendMail.body = body;
  sendMail.send();
}

function getUserAddress(ci, userId) {
  var userData = ix.checkoutUsers(ci, [userId], CheckoutUsersC.BY_IDS_RAW, LockC.NO);
  var mail = userData[0].userProps[UserInfoC.PROP_NAME_EMAIL];
  log.debug("Mail address of user " + userId + " is " + mail);

  return mail;
}

function getNodeById(workflow, nodeId) {
  var nodes = workflow.nodes;
  for (var i = 0; i < nodes.length; i++) {
    var node = nodes[i];
    if (node.id == nodeId) {
      return node;
    }
  }

  return null;
}
```

Der Empfänger erhält eine Mail mit einem klickbaren Link, der ihn in der ELO Aufgabenansicht direkt zu dem Termin führt.



Abbildung 62: Benachrichtigung im Outlook

WebApps

LISTEN IN ELO WEB APPS

AngularJS stellt einfache Möglichkeiten zur Anzeige von Listen in einer Web Oberfläche zur Verfügung. Im Prinzip muss man nur einen einzelnen Eintrag gestalten und packt diesen dann in ein DIV Tag mit der Kennung ng-repeat.

```
<div ng-repeat="picture in pictures.list">
  <div class="thumbnail">
    <a ng-href="elodms://{{picture.guid}}">
      <image ng-src="{{picture.url}}">
      <h3>{{picture.name}}</h3>
    </a>
  </div>
</div>
```

Dieses Beispiel zeigt auch gleich, wie man aus einer Web App einen Aufruf in den Java Client auslösen kann: elodms://{guid des Eintrags}

Das funktioniert bei Aufgaben auch mit der Workflow.Knoten-Id:

<a ng-href="elodms://wf/{{flow.flowId}}/{{flow.nodeId}}" ng-click="$event.cancelBubble = true">

INDEXSERVERAUFRUFE ÜBER DIE JSON API

In den WebApps und den Workflow Formularen kann man direkte Indexserverzugriffe über die JSON API ausführen. Das ist sehr praktisch, weil man nicht für jede Funktionserweiterung eine neue Serverversion ausrollen muss.

Allerdings unterscheidet sich die Arbeitsweise bei der JSON API deutlich vom normalen Indexserver Zugriff. Der JSON Zugriff kann synchron oder asynchron erfolgen – die erste Variante ist einfacher aber nicht praxisgerecht und kann leicht zu Störungen führen. Diese vergessen Sie bitte gleich wieder. Es bleibt der asynchrone Zugriff. In diesem Fall wartet das Skript bei einem Indexserverbefehl nicht auf die Antwort, sondern läuft gleich weiter. Für die Antwort wird eine Callback Funktion mitgegeben, die automatisch aufgerufen wird, wenn der Indexserver die Daten liefert.

Der große Vorteil bei dieser Arbeitsweise ist, dass das User Interface flüssig bleibt und nicht ständig kurz einfriert, wenn größere Datenmengen über eine langsame Leitung übertragen werden müssen. Der Nachteil aus der Sicht des Entwicklers ist, dass der Programmfluss schwerer zu sehen ist, weil er in eine Reihe von unabhängigen Funktionen zerfällt.

An dieser Stelle soll ein Konzept vorgeschlagen werden, welches die asynchronen Aufrufe verwendet, gleichzeitig aber optisch den Programmfluss erhält. Zu diesem Zweck werden die einzelnen Callback-Funktionen in ein Objekt eingebettet, welches nach außen die Gesamtfunktion repräsentiert.

Das folgende Beispiel ist aus dem WebApp Beispiel UserManager entnommen und soll nur das Konzept darstellen. Es ist nicht für sich alleine ausführbar.

Nach außen hin ist nur die Gesamtfunktion createTasksFolder sichtbar. Sie bekommt als Parameter das aktuelle User Objekt sowie jeweils eine Callback-Funktion für einen erfolgreichen Abschluss der Operation und einen Fehler.

Da die Funktion insgesamt vier Indexserver-Aufrufe enthält, zerfällt sie intern in entsprechend viele Teilfunktionen: checkoutRoot, doCreateSord, doSaveSord und doFinalizeSord. Diese sind so angeordnet, dass optisch der Ablauf sichtbar bleibt, auch wenn tatsächlich jede Funktion einzeln als Callback aufgerufen wird.

Im Prinzip endet jede Teilfunktion mit einem Indexserver-Aufruf. Dieser bekommt ein AsyncCallback Objekt mit der nächsten Teilfunktion, die ausgeführt wird, sobald die Ergebnisse vorliegen. Es ist eine ganz andere Programmiermethodik, an die man sich gewöhnen muss. Im Web Umfeld (nicht nur bei ELO) führt aber kein Weg daran vorbei.

```
  /**
   * Asynchroner Aufruf zum Erzeugen des User Tasks Ordners.
   *
   * @param {Model} user
   * @param {Function} okResult
   * @param {Function} errorResult
   * @returns {undefined}
   */
  createTasksFolder: function(user, okResult, errorResult) {
    var checkoutRoot = function() {
      // Schritt 1: Basisordner "Users" ermitteln
      var cb = new de.elo.ix.client.AsyncCallback(doCreateSord, errorResult);
      var path = "ARCPATH[" + user.parentId + "]:/Users";
      elo.IX.ix().checkoutSord(path, elo.CONST.EDIT_INFO.mbSord, elo.CONST.LOCK.NO,
cb);
    };

    var doCreateSord = function(parentFolder) {
      // Schritt 2: Sord Objekt Anwenderordner erzeugen
```

```
    var cb = new de.elo.ix.client.AsyncCallback(doSaveSord, errorResult);
    elo.IX.ix().createSord(parentFolder.sord.id, "1", elo.CONST.EDIT_INFO.mbSord, cb);
  };

  var doSaveSord = function(destinationFolder) {
    // Schritt 3: Anwenderordner konfigurieren und speichern
    var sord = destinationFolder.sord;
    var userid = elo.IX.getLoginResult().user.id;
    sord.name = "User." + userid;
    var acl = [new de.elo.ix.client.AclItem(31, 1, null, 0), new de.elo.ix.client.AclItem(17,
userid, null, 0)];
    sord.aclItems = acl;
    user.parentInfo = new de.elo.ix.client.EditInfo();
    user.parentInfo.sord = sord;
    var cb = new de.elo.ix.client.AsyncCallback(doFinalizeSord, errorResult);
    elo.IX.ix().checkinSord(sord, elo.CONST.SORD.mbAll, elo.CONST.LOCK.NO, cb);
  };

  var doFinalizeSord = function(id) {
    // Schritt 4: Ordner-Id eintragen
    user.parentInfo.sord.id = id;
    okResult(user.parentInfo);
  };

  checkoutRoot();
},
```

KONFIGURATIONSVARIABLEN IM SCOPE EINER APP

AngularJS stellt jedem Aufruf einen scope zur Verfügung, der als Model für die aktuellen Daten dient. Oder anders formuliert: Hier speichert man seine eigenen Werte. Allerdings sollte man nicht alle eigenen Variablen direkt in den Scope legen, sondern ein eigenes Objekt für diese Variablen anlegen.

Das hat zwei Gründe: Zum einen vermeidet man Kollisionen durch zufällige Namensgleichheit für unterschiedliche Werte in verschiedenen Programmteilen. Speziell beim Einsatz von Libraries ist es dem Entwickler nicht klar, welche Namen innerhalb der Libray verwendet werden. Andererseits führt die direkte Ablage zu unerwarteten Problemen bei vererbten scopes. Eine direkte Ablage der Variablen im scope führt dazu, dass sie nur in dem, möglicherweise temporären, vererbten scope vorhanden ist und

nicht im Basis-scope. Der Entwickler wundert sich, warum der Wert zwar gespeichert wird, kurz darauf aber nicht mehr vorhanden ist. Die Verwendung des Objekts vermeidet dieses Problem, welches durch die asynchrone Abarbeitung der Callback Funktionen noch verschärft wird.

```
myApp.controller('PictureCtrl', function($scope) {
  $scope.store = {};

  $scope.getChildren = function(result) {
    $scope.store.parentInfo = result;
    $scope.store.parentId = result.sord.id;
…
```

DRAG & DROP

Die ELO Web Apps bieten von Haus aus Unterstützung für Drag & Drop von Einträgen an. Ein Eintrag, der per D&D verschoben werden kann, muss das Kennzeichen ui-draggable="true" besitzen. Wenn man eine Rückmeldung am Ende der D&D Operation benötigt, muss man noch eine Callback Funktion on-drop-success anmelden. Diese Funktion kann z.B. das Objekt aus der alten Ordnerliste löschen.

```
<div ui-draggable="true" drag="item" on-drop-success="dropSuccessHandler($event, item, area)" ng-repeat="item in area.items">
```

Auf der möglichen Zielseite muss man eine Callback Funktion zur Behandlung des drop Vorgangs hinterlegen. Diese Funktion trägt z.B. das Objekt in die neue Ordnerliste ein.

```
<div ui-on-drop="dropItemHandler($event, $data, area)">
```

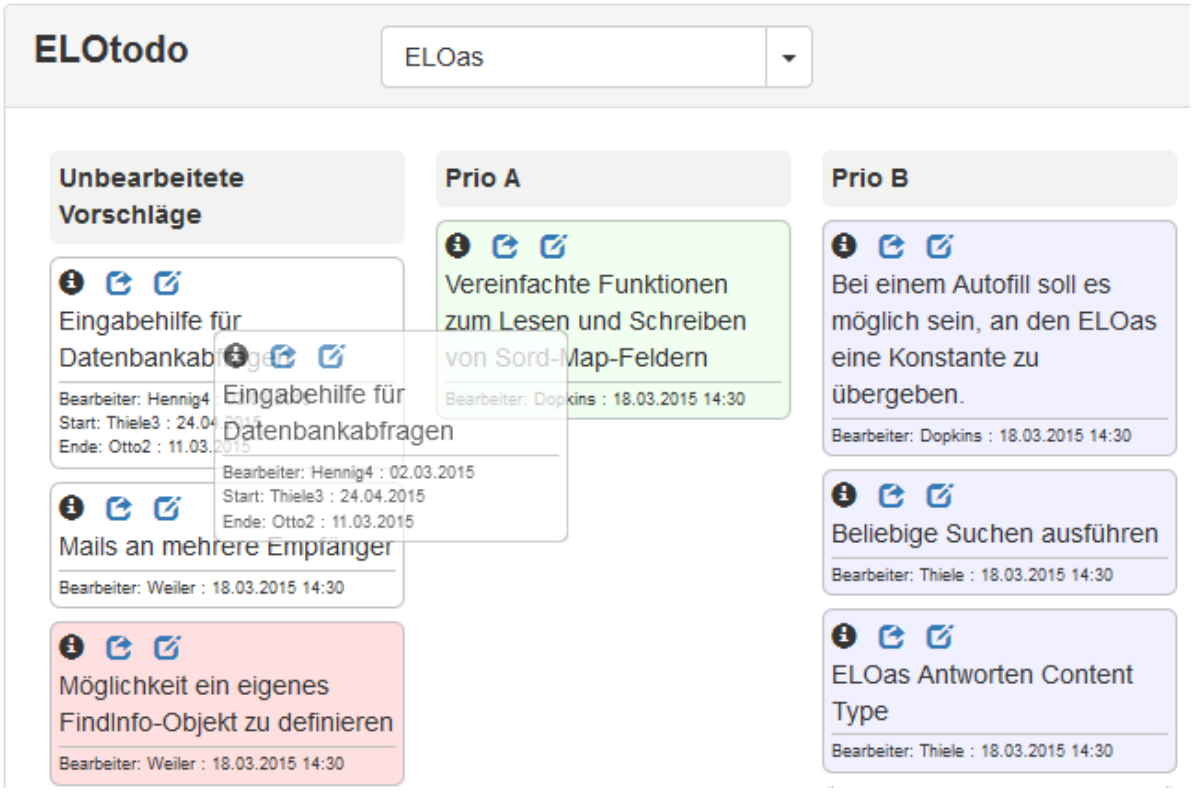


Abbildung 63: Verschieben per Drag & Drop