

ELO Indexserver

[Stand: 29.04.2016 | Programmversion: 10.00.000]

Dieses Dokument behandelt verschiedene Themen mit Bezug zum Indexserver.

Inhalt

1	Umstieg von ELO 2011 auf ELO 9	5
1.1	Konfiguration	5
1.1.1	Obsolete Optionen.....	5
1.1.2	Neue Optionen	5
1.2	Programmierung	6
1.2.1	Kompatibilität	6
1.2.2	Anpassungen in der Java-API.....	6
1.2.3	IXClient Klasse	6
1.2.4	IXConnLogin, IXConnection.getLogin().....	6
1.2.5	Anpassungen in der JSON-API.....	6
1.2.6	Anpassungen in der C# API	7
1.2.7	Anpassungen in der C++ API.....	7
2	ELO Indexserver SSO	8
2.1	Voraussetzungen	8
2.2	NTLM für Indexserver konfigurieren.....	8
2.2.1	Voraussetzungen.....	8
2.2.2	Konfiguration	8
2.3	SPNEGO konfigurieren.....	9
2.3.1	Voraussetzungen.....	9
2.3.2	SPNEGO für Linux/MacOS konfigurieren.....	9
2.3.3	Eine Kerberos Konfigurationsdatei bereitstellen	12
2.3.4	Integrierte Windows Authentifizierung auf Windows Client PCs.....	12
2.4	Client Anwendungen schreiben, die SSO über NTLM oder SPNEGO nutzen	12
2.4.1	Microsoft .NET Plattform.....	12

2.4.2	Java 7.0 Plattform	13
3	Indexserver Client Event API	14
3.1	Einführungsbeispiel: Untereinträge eines Ordners überwachen	14
3.2	Konzeptioneller Aufbau	15
3.3	Weitere Informationen	18
3.3.1	Umgang mit Ereignissen	18
3.3.2	Ereignis-Listener	19
3.3.3	Ereignisbusse	20
3.3.4	Ereignis-Handler	20
3.4	Anwendungsbeispiele	21
3.4.1	Ordner in einer Anwendung mit Benutzeroberfläche überwachen	21
3.4.2	Client-Client-Kommunikation	25
3.5	Technische Realisierung	27
4	Indexserver JSON API	28
4.1	Einführungsbeispiel: Anzeige eines Dokuments nach Objekt-ID	28
4.2	Cross-Domain Anfragen	31
4.3	Synchrone und asynchrone Aufrufe	32
4.4	Datei Upload	37
4.5	Anmeldung	41
4.5.1	Anmeldung mit Anwendername und -kennwort	41
4.5.2	SSO-Anmeldung: IXConnFactory.createSSO	41
4.5.3	Übernahme eines bestehenden Tickets	41
4.6	Elementselektoren	41
4.7	Anwendungsentwicklung, Debugging	43
4.7.1	Entwicklung mit Eclipse	43
4.8	Reverse-Proxy mit dem Apache2 Webserver	44
4.9	Beispiele	46
5	ELOix Lastverteilung	47
5.1	Überblick	47
5.2	Zusätzlichen Indexserver installieren	47
5.2.1	Konfiguration	49
5.3	Apache2 als Lastverteilung	51

5.3.1	Apache2 installieren	51
5.3.2	Apache2 konfigurieren	52
5.3.3	Tomcat Konfiguration ergänzen.....	54
5.3.4	Indexserver Option ixUrlBase setzen.....	54
5.3.5	Weitere Informationen	55
6	Zugriff auf ein ELO-Archiv über WebDAV	56
6.1	ELO WebDAV –Server-Adresse.....	56
6.1.1	Verbindung mit vollqualifiziertem Domänennamen.....	56
6.2	ELO WebDAV unter Windows verwenden.....	58
6.2.1	Mit Windows Explorer als Netzlaufwerk verbinden (ab Windows Vista)	58
6.2.2	Mit Windows XP	60
6.2.3	Verwendung des WebDAV Redirectors	61
6.2.4	Mit der Kommandozeile	61
6.2.5	Mit NetDrive.....	62
6.2.6	Mit BitKinex.....	63
6.2.7	WebDAV unter Microsoft Office Anwendungen verwenden.....	64
6.2.7.1	Objekte aus Archiv in MS Office Anwendungen einbetten.....	65
6.3	WebDAV unter Linux verwenden.....	65
6.3.1	Davfs2 (Kommandozeile)	65
6.3.2	Gnome/Nautilus	66
6.3.3	KDE/Dolphin.....	66
6.4	WebDAV unter MAC OS X verwenden	66
6.4.1	Kommandozeile.....	66
6.4.2	Finder	67
6.4.3	Cyberduck	67
6.5	WebDAV unter Android verwenden	68
6.5.1	WebDAV File Manager.....	68
6.5.2	WebDAV Nav Lite	68
6.6	Webdav unter iOS verwenden	68
6.6.1	WebDAV Navigator.....	68
6.6.2	Over The Air	68
6.7	Übersicht: WebDAV-Clients	69

7	Dynamische Stichwortlisten.....	71
7.1	Beispiel.....	71
7.2	Administration	71
7.3	Client-Anwendung	71
7.4	Beispiel-Skript.....	72
7.4.1	SampleScript.js	72
7.5	Konfiguration in der Verschlagwortungsmaske	75
7.6	Dynamische Stichwortlisten im ELOWf.....	75
7.6.1	Verwendung aus Maskendefinition wie in anderen Clients.....	75
7.6.2	Verwendung von dynamischen Stichwortlisten für Map-Felder.....	76
8	OCR über den Indexserver.....	78
8.1	Konfiguration.....	78
8.2	Aufruf der processOcr Funktion.....	79
8.2.1	Feststellen, ob OCR möglich ist	79
8.3	Landessprachen ausgeben.....	79
8.4	OCR-Analyse synchron durchführen.....	80
8.5	OCR-Analyse asynchron ausführen	80
8.6	OCR-Analyse mit Rückgabe der Zeichenpositionen	82
8.7	OCR-Analyse eines Rechtecks.....	84
9	Preview-Generierung	86
9.1	Technischer Hintergrund.....	86
9.2	Preview erzeugen.....	87
9.3	Administration	88
9.3.1	Kontrolle des Externen Prozesses	88
9.3.2	Cache.....	89
10	Skriptentwicklung	90
10.1	IXConnection Klasse in Ereignisskripten	90
11	Feed-Einträge erstellen	93

1 Umstieg von ELO 2011 auf ELO 9

1.1 Konfiguration

1.1.1 Obsolete Optionen

Option	Bedeutung
directDMAccess	<p>Indexserver gibt URLs an die Client-Anwendung, die direkt auf den ELOdm verweisen. Dadurch konnte der Dokumentenzugriff beschleunigt werden.</p> <p>Mit ELO 9 adressieren URLs immer den Indexserver.</p> <p>Die Option wurde entfernt, weil sie oft unpassend eingestellt war und Probleme verursachte.</p>
genSoundsLikeInfo	<p>Bei gesetzter Option werden für Kurzbezeichnung und Indexwerte „Sounds-Like“ Daten generiert.</p> <p>In ELO 9 werden die Informationen nicht generiert.</p> <p>Weil die „Sounds-Like“ Suche in der Praxis nicht eingesetzt wurde und durch iSearch besser abgedeckt ist, wurde die Option entfernt.</p>

1.1.2 Neue Optionen

Eine detaillierte Beschreibung zu den neuen Optionen findet man unter dem Link „Help“ auf der Optionen-Seite des Indexservers.

Option	Bedeutung
logMonitor	<p>Es wird alle 10s eine Zeile in die Log-Datei des Indexservers oder in eine separate Log-Datei geschrieben, die u.a. Informationen über Anzahl der Sitzungen, Anzahl der DB-Verbindungen, Anzahl der Threads und Speicherverbrauch beinhaltet.</p>
imagingImageCacheDir, imagingImage...	<p>Preview-Erstellung für das Rendern von Annotationen auf Grafikformate und PDFs, s. Abschnitt 9.</p>
verifyBackendTimeoutSeconds	<p>ELOam und die DB-Verbindungen werden alle 10s überwacht. Ihre erlaubte maximale Reaktionszeit kann mit</p>

	dieser Option festgelegt werden.
--	----------------------------------

1.2 Programmierung

Die Kommunikationsschicht im Indexserver wurde komplett überarbeitet.

1.2.1 Kompatibilität

Bestehende Anwendungen, die mit den Client-Libs von ELO 7.0 und ELO 2011 arbeiten, können ohne Änderung weiterhin mit dem Indexserver 9.0 kommunizieren. Im Indexserver bleibt die vorige Serialisierung – und auch SOAP – weiterhin verfügbar. Allerdings bleiben sie auf dem Stand von ELO 2011 eingefroren.

Umgekehrt werden Anwendungen, welche die neuen Client-Libs des Indexserver 9.0 einbinden, **NICHT** mit älteren Indexservern reden können.

Die neuen Client-Libs lassen sich i.d.R. ohne Änderungen oder mit sehr wenigen Änderungen in den bisherigen Client-Anwendungen verwenden.

1.2.2 Anpassungen in der Java-API

Einige öffentlichen Klassen und Funktionen der Indexserverschnittstelle stehen teilweise nur noch in dem Umfang zur Verfügung, wie sie im Indexserver Programmierhandbuch verwendet wurden.

1.2.3 IXClient Klasse

Diese Klasse steht nur noch eingeschränkt zur Verfügung. Verfügbar sind weiterhin die Subklasse „ContentStream“ und die Hilfsfunktion „getFileExt“.

1.2.4 IXConnLogin, IXConnection.getLogin()

Wurde inklusive der Subklassen entfernt. Die von der Klasse bereitgestellten Informationen können aus IXConnection.getLoginResult() ermittelt werden.

1.2.5 Anpassungen in der JSON-API

Die Zusammenstellung der Elementselektoren hat sich geändert. Alle Stellen, an denen nicht die Elementselektoren aus dem Konstantenobjekt conn.getCONST() genommen werden, müssen überarbeitet werden.

Elementselektoren waren bisher ein Array aus 4 Integer-Werten. Sie werden nun intern als Strings abgebildet und können mit dem „+“ Operator zusammenaddiert werden.

```
var sordC = CONST.SORD.mbMinMembers + CONST.SORD.mbDocVersion;  
var sordZ = new IX.SordZ(sordC);  
var editZ = new IX.EditInfoZ(0, sordZ);
```

1.2.6 Anpassungen in der C# API

Die ELO 9 API wird nur noch für .NET 4.0 bereitgestellt.

1.2.7 Anpassungen in der C++ API

Die C++ API gibt es weiterhin auf Anfrage. Es wird mindestens Microsoft Visual Studio 2012 benötigt (C++11 Standard).

2 ELO Indexserver SSO

Anwendungen, die mit Single Sign On (Automatische Systemanmeldung) arbeiten, nutzen die im Betriebssystem bekannte Identität des Benutzers zur Authentifizierung. Der Vorteil dabei ist, dass sich Anwender nicht explizit an der Anwendung anmelden müssen.

Über eine HTTP Verbindung kann SSO mittels NTLM oder SPNEGO ausgeführt werden. Das ELO System setzt dafür voraus, dass die Benutzer im Betriebssystem und in ELO mit dem gleichen Benutzernamen konfiguriert sind - das Passwort kann unterschiedlich sein.

Zur Unterstützung der SSO Funktion ist in der Indexserver WAR Datei die WAFFLE Bibliothek eingebaut. Sie funktioniert nur unter Windows. Um SSO auf anderen Betriebssystemen abbilden zu können, muss im Deployment-Deskriptor der WAR Datei ein entsprechender Servlet-Filter eingesetzt werden.

2.1 Voraussetzungen

- Indexserver 9.00.010
- EloixClient.jar bzw. EloixClientCS.dll aus Indexserver_Programming 9.00.010

2.2 NTLM für Indexserver konfigurieren

2.2.1 Voraussetzungen

- Der Server muss auf einem Windows Betriebssystem laufen.
- Tomcat muss auf einem lokalen Systemkonto laufen, oder auf einem Systemkonto mit SeTcbPrivilege "Als Teil des Betriebssystems handeln".
- Auf dem Server muss Java Runtime 7.0 oder höher vorhanden sein.
- Java Client Anwendungen müssen auf Java 7.0 oder höher laufen.

2.2.2 Konfiguration

- Wenn sich nur Benutzer einer bestimmten Domäne einloggen sollen, fügen Sie in der <instdir>/config/ix-Archivname/config.xml des Indexservers diese Zeile ein:

```
<entry key="ntlm.domain">case-insensitive-domain-name</entry>
```

In diesem Fall enthält die NT-Username property in der ELO-Benutzerverwaltung nur den Benutzernamen, nicht den Domänen Namen.

- Wenn sich Benutzer aus unterschiedlichen Domänen anmelden, fügen Sie diese Zeile ein:

```
<entry key="ntlm.domainUserFormat">format-specification </entry>
```

Die NT-Username property muss in diesem Fall den Domänen Namen und den Benutzernamen enthalten, formatiert in der entsprechenden Formatspezifikation. Die Formatspezifikation ist ein String, in dem der Platzhalter {0} für den Domännennamen steht und der Platzhalter {1} für den Anwendernamen. Beispiel: {0}\{1} ergibt Domäne\Anwender.

Tomcat neu starten

Für Windows 2003 Clients fügen Sie den Servernamen zu den vertrauenswürdigen Seiten im lokalen Internetbereich hinzu.

2.2.2.1 Konfiguration verifizieren

Öffnen Sie den Internet Explorer und navigieren Sie zur Indexserver Statusseite. Klicken Sie dort den Link „Test SSO Login“.

Der Browser sollte nicht nach Name und Passwort fragen. Das Ergebnis sollte aussehen wie in: Abbildung 1, Testausgabe.

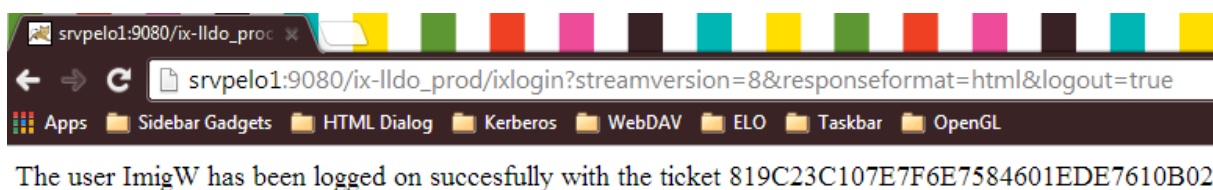


Abb. 1: Testausgabe

2.3 SPNEGO konfigurieren

SPNEGO verwendet das Kerberos Protokoll, das in den letzten Jahren zu einem Branchenstandard geworden ist.

2.3.1 Voraussetzungen

- Auf dem Server muss Java Runtime 7.0 oder höher vorhanden sein.
- Java Client Anwendungen müssen auf Java 7.0 oder höher laufen.

Zuerst muss im Key Distribution Center (KDC) ein Kerberos-Prinzipal für den HTTP Service erstellt werden.

In der folgenden Beschreibung wird für den Server, auf dem Indexserver oder Tomcat laufen, der Servername SRVT02 verwendet. Das Passwort für den Benutzer (z.B.: krb_SRVT02), der mit dem Prinzipal assoziiert wird, ist "elo". Dieser Benutzer muss Rechte für Kerberos Delegation haben: `setspn -A HTTP/SRVT02.ELO.LOCAL krb_SRVT02`.

Für die Kerberos Umgebung (Domänenname) wird "ELO.LOCAL" verwendet.

2.3.2 SPNEGO für Linux/MacOS konfigurieren

2.3.2.1 Kerberos Dienstprinzipalname und Keytab File java.keytab

- Legen Sie einen Domänen Benutzer an, z.B.: krb_SRVT02

- Mit Hilfe des ktpass command aus dem Windows Resource Kit mappen Sie den Dienstprinzipalnamen auf den Domänenbenutzer und generieren ein Keytab File. Diese Datei enthält einen privaten Schlüssel, um eingehende Anfragen zu verschlüsseln und sollte für niemanden außer für den Administrator und das Tomcat Service Konto sichtbar sein.

```
ktpass -princ HTTP/SRVT02.ELO.LOCAL@ELO.LOCAL  
-pass elo -mapuser krb_SRVT02@ELO.LOCAL  
-ptype KRB5_NT_PRINCIPAL  
-out /etc/java.keytab  
-mapOp set
```



- **Hinweis:**
- Benutzen Sie im Domänennamen nur GROSSBUCHSTABEN.
- Die Datei können Sie z.B. hier ablegen:

```
<eloserverinstdir>/config/Kerberos
```

2.3.2.2 Kerberos Konfigurationsdatei für Java bereitstellen: jaas-krb5.conf

Um Kerberos mit Java zu nutzen, müssen Sie für den JAAS der Java Runtime eine Konfigurationsdatei bereitstellen. Der Indexserver benötigt eine Konfigurationsdatei, die folgendermaßen aussieht:

```
IndexServer {  
  com.sun.security.auth.module.Krb5LoginModule required  
  debug=false  
  useKeyTab=true  
  storeKey=true  
  keyTab="g:/ELOprofessional/config/kerberos/java.keytab"  
  principal="http/SRVT02.ELO.LOCAL@ELO.LOCAL" ;  
};
```



- **Hinweis:**
- Benutzen Sie im Domänennamen nur GROSSBUCHSTABEN.
- Die Datei können Sie z.B. hier ablegen:

```
<eloserverinstdir>/config/Kerberos/jaas-krb5.conf
```

- Der Kerberos Layer der Java VM schreibt umfangreiche Informationen in die stdout* log Datei, wenn der debug Parameter auf true eingestellt ist.

2.3.2.3 SPNEGO für Indexserver konfigurieren

Die Indexserver Konfiguration in der config.xml Datei muss um ein paar Einträge ergänzt werden. Die config.xml liegt standardmäßig im Verzeichnis

```
<eloserverinstadir>/config/ix-<archivename>.
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment>parameters for this web application</comment>
<entry key="jdbcurl">jdbc:sqlserver://srvt02:1433</entry>
<entry key="dbdriver">com.microsoft.sqlserver.jdbc.SQLServerDriver</entry>
<entry key="dbpwd">81-41-181-112-57-23-24-141</entry>
<entry key="schema"/>
<entry key="database">elo70</entry>
<entry key="dbuser">elodb</entry>
<entry key="kerberos.jaas.conf">g:/ELOprofessional/config/jaas-
krb5.conf</entry>
<entry key="kerberos.login">IndexServer</entry>
<entry key="kerberos.kdc">NEGRIL.ELO.LOCAL</entry>
<entry key="kerberos.realm">ELO.LOCAL</entry>
</properties>
```

Property Name	Description
kerberos.jaas.conf	Ablageort der jaas-krb5.conf-Datei. Obligatorisch.
kerberos.login	Dieser Wert entspricht dem ersten Symbol in der jaas-krb5.conf-Datei. Optional, Voreinstellungen zum Indexserver.
kerberos.kdc	Computernamen des Kerberos Key Distribution Center (Active Directory Server). Optional, siehe 6.4.5. <i>Eine Kerberos Konfigurationsdatei bereitstellen.</i>
kerberos.realm	Kerberos Umgebung (Windows Domänenname). Optional, siehe 6.4.5. <i>Eine Kerberos Konfigurationsdatei bereitstellen.</i>

2.3.3 Eine Kerberos Konfigurationsdatei bereitstellen

Normalerweise sind das Kerberos Key Distribution Center und die Umgebung für alle (Java) Anwendungen auf einem Computer die gleichen. Das heißt, es ist praktischer, diese Eigenschaften in einer global verfügbaren Datei zu spezifizieren anstatt für jede einzelne Anwendung.

Java Runtime akzeptiert Kerberos Konfigurationsparameter in einer Datei mit dem Namen krb5.ini im %SystemRoot% directory. (krb5.conf in nicht-Windows Betriebssystemen).

```
The file looks like this:
[libdefaults]
    default_realm = ELO.LOCAL
[realms]
    ELO.LOCAL = {
        kdc = NEGRIL.ELO.LOCAL
    }
```

2.3.4 Integrierte Windows Authentifizierung auf Windows Client PCs

Um Kerberos Authentifizierung für HTTP (SPNEGO) für Windows Clients zu nutzen, folgen Sie den Schritten in folgender Dokumentation: <http://technet.microsoft.com/en-us/library/cc779070.aspx>

Die Webseite die hier hinzugefügt wird, sollte so aussehen: <http://SRVT02.ELO.LOCAL>

2.3.4.1 Die Konfiguration verifizieren

Wie in 2.2.2.1.



- **Hinweis:** Nutzen Sie immer den Kerberos-Dienstprinzipalnamen als Servernamen in der URL: SRVT02.ELO.LOCAL anstelle von SRVT02. Ansonsten misslingt die Authentifizierung und die Debug-Ausgabe im stdout*.log zeigt einen "Checksum failed" Fehler.

2.4 Client Anwendungen schreiben, die SSO über NTLM oder SPNEGO nutzen

2.4.1 Microsoft .NET Plattform

Das folgende Codesegment loggt über SSO ein:

```
ClientInfo ci = ...
IXConnFactory connFact = ...
IXConnection ix = connFact.CreateSso(ci, "mycomputer");
```

2.4.1.1 .NET auf Windows Vista, Windows 7, Windows 2008

Weil .NET 2.0 nicht das aktuellste Authentifizierungsprotokoll unterstützt, setzen Sie den Registry-Schlüssel auf:

```
HKLM\SYSTEM\CurrentControlSet\Control\Lsa\LMCompatibilityLevel=1
```

<http://technet.microsoft.com/en-us/library/cc960646.aspx>

2.4.2 Java 7.0 Plattform

Das folgende Codesegment loggt über SSO ein:

```
ClientInfo ci = ...  
IXConnFactory connFact = ...  
IXConnection ix = connFact.createSso(ci, "mycomputer");
```

3 Indexserver Client Event API

Der Indexserver erlaubt bereits seit der Version 7.00.020 die Behandlung von Ereignissen (z. B. Einchecken eines Dokuments) in JavaScript-Funktionen oder Webservices. Bisher konnte die Ereignisverarbeitung aber nur auf dem Server erfolgen. Ab der Version 8.00.010 ist es möglich, auf bestimmte Ereignisse (z. B. Änderung der Untereinträge eines Ordners) in der Clientanwendung zu reagieren. Darüber hinaus können sich Clientanwendungen untereinander in einer Art Instant-Messaging selbstdefinierte Ereignisse zusenden.

3.1 Einführungsbeispiel: Untereinträge eines Ordners überwachen

Das folgende Beispiel zeigt, wie ein Ordnerinhalt überwacht werden kann. Werden Untereinträge hinzugefügt, gelöscht oder bearbeitet, erhält die Clientanwendung ein Ereignis.

Voraussetzung für die Abarbeitung von Ereignissen ist, dass die Clientanwendung eine Schnittstelle bereitstellt, an die die Ereignisse übergeben werden können. In .NET Anwendungen geschieht dies in Form eines Delegate-Objekts, das an die Event-Eigenschaft „EventBusHandler“ angehängt wird.

Anschließend kann durch einen oder mehrere „EventListener“ beschrieben werden, welche Ereignisse für die Clientanwendung von Interesse sind. Die Funktion „AddListener“ richtet hier einen „EventListener“ für den Ereignistyp „EventBusC.EVENT_TYPE_WATCH_FOLDER“ ein. Zu diesem Ereignistyp muss noch angegeben werden, welcher Ordner überwacht werden soll. Dies geschieht durch Übergabe von „folderId“.

```
public static void WatchFolder()
{
    IXConnection conn = ...

    conn.EventBusApi.EventBusHandler += new
EventBusApi.ProcessEventBusEvents(
EventBusApi_EventBusHandler);

    // This folder is being watched
    String folderId = "1";

    // Add event listener for watching the folder

conn.EventBusApi.BroadcastBus.AddListener(EventBusC.EVENT_TYPE_WATCH_FOLDER
, 0,

                                     folderId);

    // Process events for 1min
    Thread.Sleep(60 * 1000);

    // Delete all attached event listeners
    conn.EventBusApi.BroadcastBus.Release();
}
```

```
}  
  
static void EventBusApi_EventBusHandler(long subsId, Event[] events)  
{  
    foreach (Event e in events)  
    {  
        Object[] args = (Object[])AnyToObject.ToObject(e.any);  
        Sord sord = (Sord)args[0];  
        SordZ sordZ = (SordZ)args[1];  
        int what = (int)args[2];  
  
        Console.WriteLine("Event: " + e.type +  
            ", folderId=" + e.param2 +  
            ", modified sord=" + sord.name +  
            ", what=" + what);  
    }  
}
```

Wenn das Programm gestartet wird, werden für eine Minute die Änderungsereignisse der obersten Archivstruktur ausgegeben.

3.2 Konzeptioneller Aufbau

Im Zentrum der Ereignisweiterleitung stehen die Ereignisbusse („Event Busses“), an die Ereignisse zur Verteilung gesendet werden (s. Abbildung 4).

An einem Ereignisbus können die Teilnehmer („Subscribers“) Ereignisse abhören. Ein Teilnehmer ist in der Regel gleichbedeutend mit einer Sitzung. Teilnehmer können auch Ereignisse an einen Ereignisbus senden, z. B. um mit anderen Teilnehmern Nachrichten auszutauschen. Des Weiteren werden Ereignisse von Indexserver-Funktionen wie z. B. `checkinSord` erzeugt.

Ein Teilnehmer kann einen vordefinierten Ereignisbus oder einen selbst erstellten für die Kommunikation nutzen. Zu den vordefinierten Ereignisbussen gehört der „Broadcast Event Bus“, der eine uneingeschränkte Kommunikation der Teilnehmer untereinander erlaubt. Daneben stellt der Indexserver je Anwender einen Ereignisbus bereit, der wie ein Postfach funktioniert: nur der Eigentümer kann Ereignisse abhören, andere können nur Ereignisse senden.

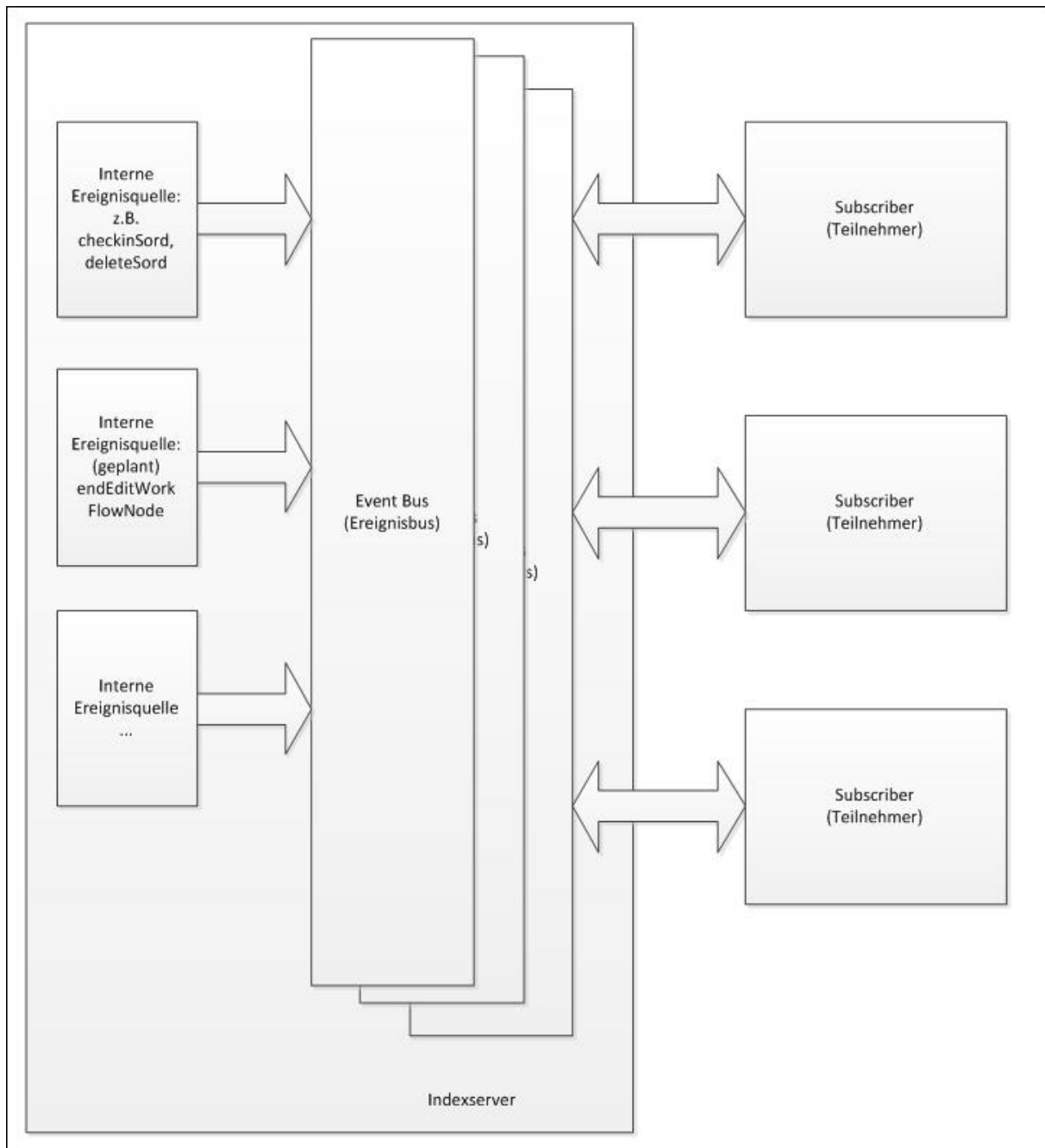


Abb. 2: Ereignisbus und Teilnehmer

Damit die Clientanwendung Ereignisse behandeln kann, muss sie zunächst einen „EventBusHandler“ bereitstellen. In Java-Programmen geschieht dies durch die Übergabe eines Objekts, das die Schnittstelle „IXEventBusHandler“ implementiert, an die Funktion „IXConnection.EventBusApi.setHandler“. In .NET-Programmen ist der „EventBusHandler“ ein Delegate-Objekt, das an das Ereignis „IXConnection.EventBusApi.EventBusHandler“ gehängt wird.

Mit dem Zuweisen des „EventBusHandler“ wird in der „IXConnection“ zugleich auch ein Teilnehmer erstellt.

Anschließend können „EventListener“ eingerichtet werden, deren „EventFilter“ die gewünschten Ereignisse aus den Bussen abhören und an die Clientanwendung weitergeben. „EventListener“ können zu jeder Zeit eingerichtet und gelöscht werden. In der Abbildung 5 sind die Sachverhalte noch einmal grafisch dargestellt.

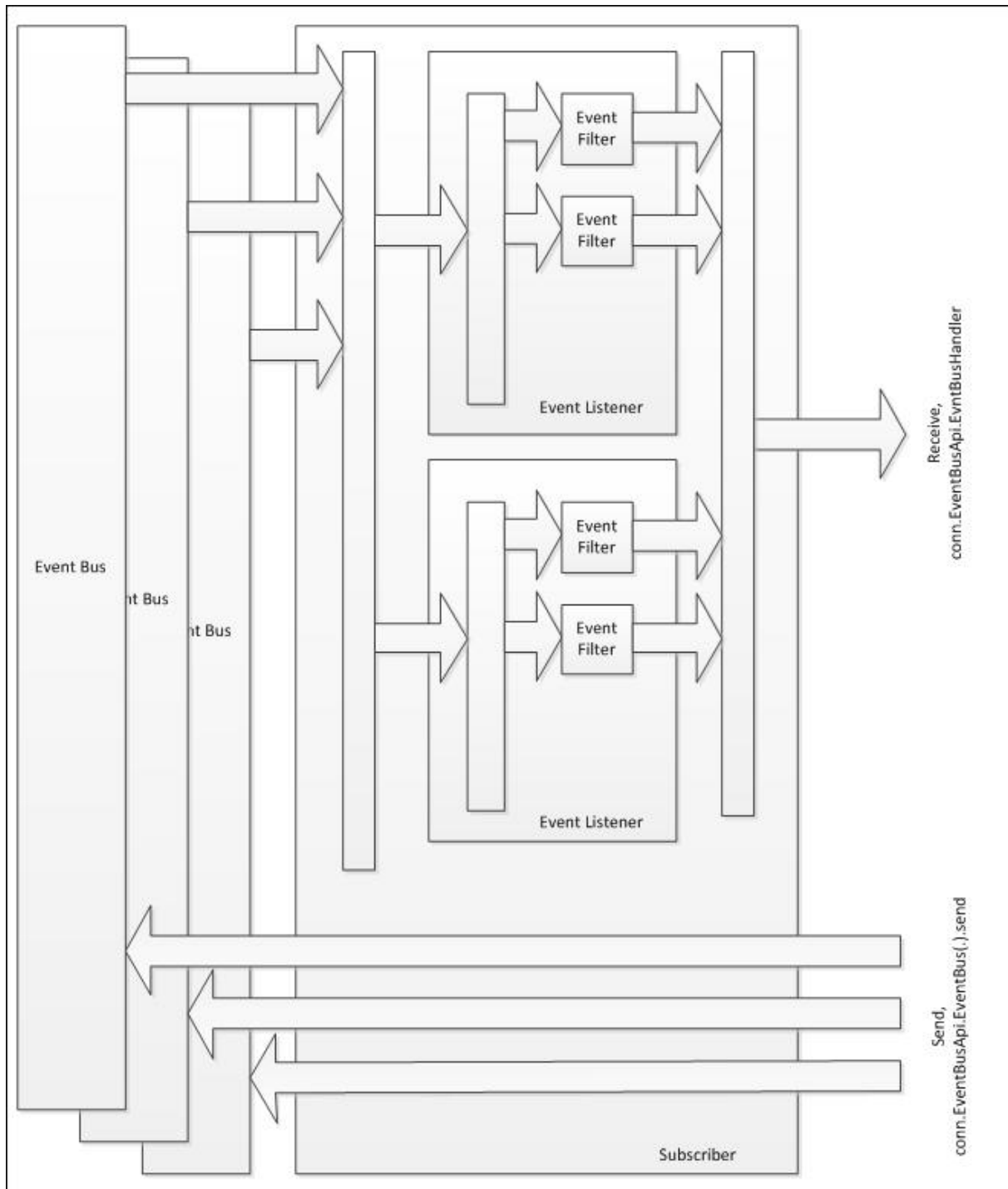


Abb. 3: Teilnehmer, Ereignis-Listener und -Filter

Über die Ereignisse können Nachrichten vom Typ „Any“ übertragen werden. Dieser Typ ist in der Lage, die Standardtypen wie int, long, String usw. aufzunehmen. Darüber hinaus können auch alle API-Klassen der Indexserver-Schnittstelle in ein Any eingepackt werden, z. B. Sord oder DocMask.

3.3 Weitere Informationen

3.3.1 Umgang mit Ereignissen

3.3.1.1 Ereignistypen

Es gibt einige vordefinierte Ereignistypen, deren Werte als Konstanten in EventBusC definiert sind. Darüber hinaus können Anwendungen ihre eigenen Ereignistypen definieren. Ihre Werte müssen größer als EventBusC.EVENT_TYPE_MAX_SYSTEM sein. Ereignistypen mit kleineren Werten sind für ELO reserviert.

Um Überschneidungen der Werte verschiedener Anwendungen zu vermeiden, sollten sie mit einem 64bit Zufallszahlengenerator gebildet werden.

In .NET Programmen kann man wie folgt vorgehen:

```
long computeMyEventType()  
{  
    Random rand = new Random();  
    long type = 0;  
  
    while (type <= EventBusC.EVENT_TYPE_MAX_SYSTEM)  
    {  
        byte[] buf = new byte[8];  
        rand.NextBytes(buf);  
        type = BitConverter.ToInt64(buf, 0);  
        if (type < 0) type = -type;  
    }  
    return type;  
}
```

In Java-Programmen wäre der folgende Code geeignet:

```
public static long createMyEventType() {  
    Random rand = new Random();  
    long type = 0;  
  
    while (type <= EventBusC.EVENT_TYPE_MAX_SYSTEM)  
    {  
        byte[] buf = new byte[8];  
        rand.nextBytes( buf );  
        BigInteger bint = new BigInteger(buf);  
        type = bint.longValue();  
        if (type < 0) type = -type;  
    }  
}
```

```
return type;  
}
```

Um eine private Kommunikation zwischen Client-Anwendungen aufzubauen, sollten nicht nur eigene Ereignistypen verwendet werden. Zusätzlich sollten die Ereignisse auch über einen anwendungsdefinierten Ereignisbus statt über den Broadcast-Bus laufen. Dadurch kann der Indexserver schneller die Empfänger der Ereignisse finden.

3.3.1.2 Ereignis-Parameter

Neben dem Ereignistyp können die optionalen Parameter `Event.param1` und `Event.param2` verwendet werden, um das Ereignis genauer zu beschreiben. Die Ereignisfilter im Indexserver werten neben dem Typ auch die Parameter aus, um zu entscheiden, ob ein Teilnehmer das Ereignis bekommt.

3.3.1.3 Ereignis-ACL

Einem Ereignis kann eine ACL mitgegeben werden, sodass nur leseberechtigte Teilnehmer das Ereignis empfangen.

3.3.1.4 Ereignis-IDs

Wenn Client-Anwendungen über die Ereignis-API kommunizieren wollen, dann wäre möglicherweise eine Art Request-Response-Protokoll hilfreich. Es würde ermöglichen, dass ein Sender auf eine Anfrage eine zugehörige Antwort empfangen könnte. In der Ereignis-API wird jedoch kein Request-Response-Protokoll zwischen Clients angeboten. Dies kann aber leicht selbst entwickelt werden.

Für den Sender ist es dabei wichtig herausfinden zu können, welches gesendete Ereignis zu welchem empfangenen passt. Er kann dies anhand der Ereignis-ID ermitteln, die er frei vergeben kann. Er muss sich dabei darauf verlassen können, dass der Empfänger die ID der Anfrage in die Antwort einträgt.

Die Ereignis-IDs werden vom ELOix nicht ausgewertet und können beliebige Werte annehmen.

3.3.1.5 Ereignisdaten

Die Nutzlast eines Ereignisses wird im Element `Event.any` transportiert. Any-Objekte können verschiedene Datentypen enthalten. Neben den Standardtypen wie `int`, `long`, `String` sind auch alle Klassen erlaubt, die in der Indexserver-Schnittstelle definiert sind – also z.B. auch `Sord`. Darüber hinaus können Arrays transportiert werden. Die Any-Klasse wird auch in der API der Registrierten Funktionen verwendet.

3.3.2 Ereignis-Listener

Die Ereignis-Listener fassen eine Liste von Ereignis-Filtern zusammen. Damit können mehrere Filter mit einem Aufruf eingerichtet und wieder gelöscht werden.

Listener werden automatisch gelöscht, wenn die Sitzung abgemeldet oder die Sitzungslebenszeit überschritten wird.

3.3.3 Ereignisbusse

Ereignisse können an vordefinierte Busse gesendet werden, die immer verfügbar sind und nicht explizit geöffnet werden müssen. Daneben können Anwendungen selbst Busse definieren, die sie mit der Funktion `openEventBus` öffnen und mit `closeEventBus` schließen müssen.

3.3.3.1 Broadcast Bus

Dieser vordefinierte Bus hat die ID `EventBusC.BUSID_BROADCAST` und erlaubt es jeder beliebigen Sitzung Ereignisse zu senden und abzuhören.

3.3.3.2 User Bus

Für jeden Anwender gibt es einen vordefinierten Bus, den nur er selbst abhören kann. Andere Anwender können nur Ereignisse an den Bus senden. Der Bus funktioniert also ähnlich wie ein Postfach. Die Bus-ID eines Anwenders errechnet sich aus `EventBusC.BUSID_USER + <Anwendernummer>`.

3.3.3.3 Anwendungsdefinierte Busse

Anwendungen können selbst neue Ereignisbusse öffnen. Dies ist beispielsweise dann interessant, wenn eine Anwendung aus verschiedenen Prozessen besteht, die unter derselben Sitzung (Ticket) arbeiten und miteinander kommunizieren sollen. Hierfür ist ein Ticket-bezogener Ereignisbus hilfreich.

Die Funktion `openEventBus` der Indexserver API erlaubt es, einen anwenderdefinierten Bus zu öffnen. Sie nimmt als Parameter ein `EventBusParams` Objekt entgegen. Wenn zwei Anwendungen gleiche Parameterobjekte übergeben, dann erhalten sie von `openEventBus` dieselbe Bus-ID zurück.

Die mit `openEventBus` erstellten Busse müssen mit `closeEventBus` wieder geschlossen werden.

3.3.4 Ereignis-Handler

Der Aufruf der Ereignis-Handler-Funktion erfolgt in einem Hintergrund-Thread. Für .NET-Anwendungen gibt es dafür einen Thread je `IXConnection`. In Java-Anwendungen stellt `HttpAsyncClient` zwei Threads bereit, die aber für alle `IXConnection` Objekte einer `IXConnFactory` verwendet werden. Java spart also Threads durch asynchrones IO.

In beiden Umgebungen sollte die Durchführung der Ereignis-Handler-Funktion möglichst schnell von statten gehen. Solange die Funktion läuft, kann der Hintergrund-Thread keine Ereignisse empfangen. Lang dauernde Verarbeitungen (SQL-Abfragen, Webservice-/ELOix-Aufrufe) sollten an einen anderen Thread übergeben werden.

3.4 Anwendungsbeispiele

3.4.1 Ordner in einer Anwendung mit Benutzeroberfläche überwachen

In UI-Anwendungen ist in der Regel darauf zu achten, dass die Ereignis-Handler-Funktion nicht direkt die UI-Elemente aktualisieren kann, weil sie aus einem Hintergrund-Thread aufgerufen wird. Die Ereignisinformationen müssen zuerst an den UI-Thread übergeben werden.

Dafür stellen .NET und Java die nötigen Funktionen bereit. In .NET benutzt man `Control.Invoke` und übergibt ein `Delegate`-Objekt. Das Java-AWT definiert `EventQueue.invokeLater`, und Java-Swing stellt `SwingUtilities.invokeLater` bereit.

Im Folgenden wird ein .NET Beispieldialog diskutiert, indem die ID des zu überwachenden Ordners eingegeben wird. In einer Liste werden alle Ereignisse zu Änderungen der Untereinträge aufgelistet. (Das Beispiel gehört zu dem .NET-Example-Projekt der `IndexServer_Programming.zip`).

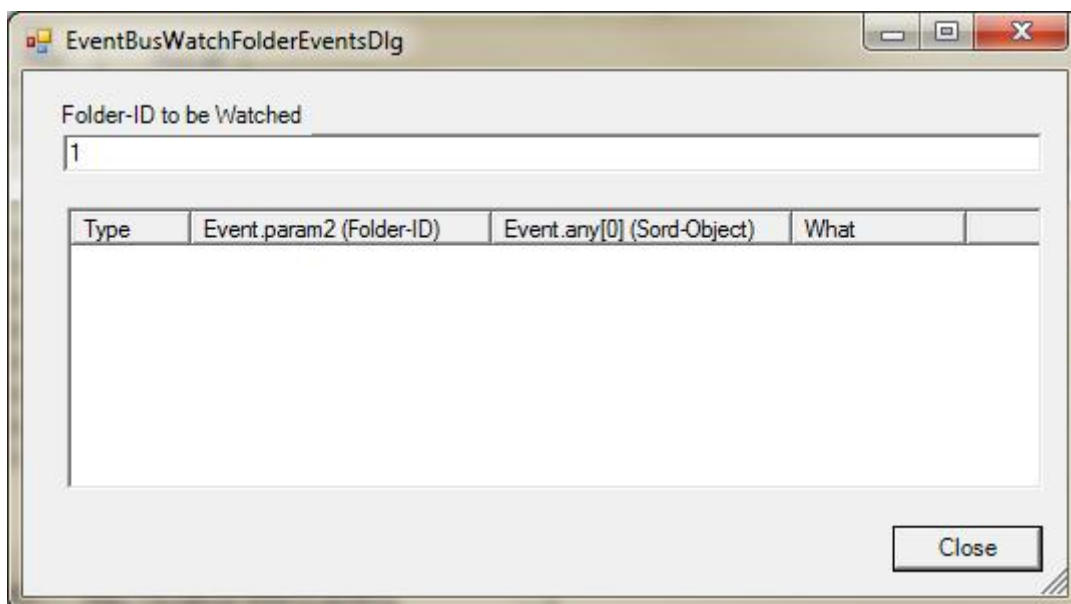


Abb. 4: *EventBusWatchFolderEventsDlg*

Der Konstruktor `EventBusWatchFolderEventsDlg(IXConnection ix)` richtet zum einen den Ereignis-Handler im `IXConnection`-Objekt ein. Zum anderen ruft er `initListener` auf, um einen Ereignis-Listener für die Ordner-Änderung zu setzen.

Die Ereignis-Handler-Funktion `EventBusApi_EventBusHandler` erstellt ein `Delegate`-Objekt vom Typ `ShowEventsCallback` mit der Funktion `ShowEvents` und übergibt es zusammen mit den Ereignissen an `Control.Invoke`. Somit wird `ShowEvents` mit der Ereignisliste im UI-Thread aufgerufen.

Ändert sich die Ordner-ID, ruft die Funktion `edObjId_TextChanged` wieder `initListener` auf, um einen neuen Ereignis-Listener für diesen Ordner zu erstellen. Zuvor wird der Ereignis-Listener auf den vorigen Ordner gelöscht.

Wird der Dialog geschlossen, so stellt `EventBusWatchFolderEventsDlg_FormClosed` sicher, dass keine weiteren Ereignisse mehr beim Dialog ankommen, indem der Ereignis-Handler abgehängt wird. Anschließend werden mit `conn.EventBusApi.BroadcastBus.Release()` noch alle Listener gelöscht, die noch mit dem Broadcast-Bus verbunden sind – hier ist es nur ein Listener, nämlich der in `initListener` eingerichtete.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using EloixClient.IndexServer;

namespace Examples
{
    /// <summary>
    /// This dialog shows the events sent by ELOix, when the
    /// subitems of a folder have changed.
    /// </summary>
    /// <remarks>
    /// The events are received from a background thread.
    /// To show the events in the list view,
    /// the control has to be called in a thread safe way.
    /// See: http://msdn.microsoft.com/en-
    us/library/ms171728\(v=vs.100\).aspx
    /// </remarks>
    public partial class EventBusWatchFolderEventsDlg : Form
    {
        // Indexserver connection
        IXConnection conn;

        // Current listener ID
        long watchFolderLsnId;

        /// <summary>
        /// This delegate type is implemented by the ShowEvents function.
        /// </summary>
        /// <param name="subsId">Subscriber ID</param>
        /// <param name="events">Events</param>
        delegate void ShowEventsCallback(long subsId, Event[] events);

        /// <summary>
        /// Constructor
    }
```

```
/// </summary>
/// <param name="conn">Indexserver connection</param>
public EventBusWatchFolderEventsDlg(IXConnection conn)
{
    this.conn = conn;

    // Create dialog UI
    InitializeComponent();

    // Add an event bus handler function to the connection.
    conn.EventBusApi.EventBusHandler +=
        new
EventBusApi.ProcessEventBusEvents(EventBusApi_EventBusHandler);

    // Add a listener for the object ID given in TextBox edObjId
    initListener();
}

/// <summary>
/// Delete the current listener, add a new listener.
/// </summary>
private void initListener()
{
    // Delete current listener
    if (watchFolderLsnId != 0L)
    {
conn.EventBusApi.BroadcastBus.DeleteListener(watchFolderLsnId);
        watchFolderLsnId = 0L;
    }

    // Add new listener
    try
    {
        watchFolderLsnId =
conn.EventBusApi.BroadcastBus.AddListener(
            EventBusC.EVENT_TYPE_WATCH_FOLDER, 0, edObjId.Text);
    }
    catch (Exception)
    {
        // Ignored.
        // initListener() is called on each change in TextBox
edObjId.

        // There might be an invalid object ID during editing.
    }
}

/// <summary>
/// Event bus handler function.
/// </summary>
/// <remarks>
```

```
/// This function is called in a background thread.
/// Thus, it cannot access the dialog elements of this form
directly.
/// We have to create a delegate object that contains the call
/// ShowEvents to update the UI.
/// This delegate is passed to the Invoke function which executes
/// ShowEvents in the UI thread.
/// </remarks>
/// <param name="subsId">Subscriber ID</param>
/// <param name="events">Events</param>
void EventBusApi_EventBusHandler(long subsId, Event[] events)
{
    ShowEventsCallback d = new ShowEventsCallback(ShowEvents);
    this.Invoke(d, new Object[] { subsId, events });
}

/// <summary>
/// This function adds the events to the list view.
/// </summary>
/// <param name="subsId">Subscriber ID</param>
/// <param name="events">Events</param>
private void ShowEvents(long subsId, Event[] events)
{
    foreach (Event e in events)
    {
        ListViewItem item = new ListViewItem();
        item.Text = Convert.ToString(e.type);
        item.SubItems.Add(e.param2);
        Object[] args = (Object[])AnyToObject.ToObject(e.any);
        Sord sord = (Sord)args[0];
        item.SubItems.Add(sord.name);
        int what = (int)args[2];
        switch (what)
        {
            case EventBusC.WATCH_INSERT:
item.SubItems.Add("inserted"); break;
            case EventBusC.WATCH_UPDATE:
item.SubItems.Add("updated"); break;
            case EventBusC.WATCH_DELETE:
item.SubItems.Add("deleted"); break;
        }
        this.lvEvents.Items.Add(item);
    }
}

/// <summary>
/// Update the listener on each change of the TextBox.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void edObjId_TextChanged(object sender, EventArgs e)
```



```
{
    initListener();
}

/// <summary>
/// Release the broadcast event bus in order to delete attached
/// listeners and detach the event bus handler.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void EventBusWatchFolderEventsDlg_FormClosed(object sender,
    FormClosedEventArgs e)
{
    try
    {
        // Detach event bus handler.
        // EventBusApi_EventBusHandler must not be called
        // after the Form has been disposed.
        conn.EventBusApi.EventBusHandler -=
            new
EventBusApi.ProcessEventBusEvents(EventBusApi_EventBusHandler);

        // Delete all of my attached listeners.
        conn.EventBusApi.BroadcastBus.Release();
    }
    catch (Exception)
    {
    }
}
}
```

3.4.2 Client-Client-Kommunikation

Es können nicht nur Ereignisse vom Server zur Clientanwendung versendet werden. Auch können Clientanwendungen untereinander Informationen austauschen.

Das folgende Beispiel simuliert eine Verbindung zwischen zwei Clients, indem zwei Sitzungen geöffnet werden. Die erste Sitzung, conn1, wartet auf ein Ereignis, die zweite sendet es. Beiden Sitzungen ist der Ereignistyp myEventType und der Kanalname channelName vorab bekannt.

```
public static void SendMessage()
{
    IXConnFactory connFact = ...

    // Create a most likely unique event ID for this communication.
    long myEventType = computeMyEventType();
```

```
// The two connection share the same event bus.
// Its bus ID is created based on this channel name.
String channelName = "de.elo.ix.tutorial.SendReceive";

// Create first connection - the receiver
IXConnection conn1 = connFact.Create("Administrator", "elo",
computerName, null);

// Add an event handler to the receiver
conn1.EventBusApi.EventBusHandler += new
EventBusApi.ProcessEventBusEvents(
    EventBusApi_EventBusHandler2);

// Open the channel.
EventBusApi.Bus channelBus1 = conn1.EventBusApi.OpenEventBusChannel(
    channelName);

// Add a listener to the channel,
// in order to receive events of type myEventType.
channelBus1.AddListener(myEventType);

// Create second connection - the sender
IXConnection conn2 = connFact.Create("Administrator", "elo",
computerName, null);

// Open the channel.
// channelBus2 uses the same bus ID as channelBus1, because it is
opened
// with the same channel name.
EventBusApi.Bus channelBus2 =
conn2.EventBusApi.OpenEventBusChannel(channelName);

// Send message
String message = "hallo conn2";
Console.WriteLine("send event: type=" + myEventType + ", message=" +
message);
channelBus2.Send(myEventType, message);

// Wait some time.
// The first connection should receive the message during this time.
Thread.Sleep(2 * 1000);

// Delete all listeners from the channel.
// A bus is deleted inside Indexserver, if the last listener is
deleted.
channelBus1.Release();

// Logout connection.
conn2.Logout();
conn1.Logout();
}
```

```
static void EventBusApi_EventBusHandler2(long subsId, Event[] events)
{
    foreach (Event e in events)
    {
        Console.WriteLine("received event: type=" + e.type +
                           ", message=" +
AnyToObject.ToObject(e.any).ToString());
    }
}
```

Dieses Beispiel stellt bereits die Basis einer Instant-Messaging-Anwendung dar, über die Anwender chatten könnten.

Neben Textnachrichten können aber auch andere Informationen versandt werden. Beispielsweise könnte von einer Arbeitsplatzanwendung ein Sord-Objekt an einen Serverprozess übermittelt werden, der damit bestimmte Aktionen durchführt.

Mit etwas mehr Aufwand kann die Kommunikation in ein Request-Response-Schema gefasst werden, sodass der Sender der Nachricht vom Empfänger eine Antwort bekommt.

3.5 Technische Realisierung

Die Kommunikation zum Indexserver erfolgt über das HTTP-Protokoll, das prinzipiell Verbindungen nur in einer Richtung – vom Client zum Server – zulässt. Für die Ereignisse, die im Server ausgelöst und im Client behandelt werden sollen, wird jedoch eigentlich die umgekehrte Richtung benötigt.

Eine Lösung könnte sein, die Clientanwendung in kurzen Abständen nach neuen Ereignissen fragen zu lassen. Der Nachteil davon ist aber, dass bei vielen Clientanwendungen auch sehr viele Aufrufe übers Netzwerk gehen und vom Indexserver verarbeitet werden müssen.

Weniger Anfragen verursacht eine Technik, die oft als „Long-Poll“ bezeichnet wird. Hierbei sendet der Client eine Anfrage, die vom Server solange nicht beantwortet wird, bis die gewünschten Daten vorliegen. Das gravierendste Problem dieser Vorgehensweise war in der Vergangenheit, dass in der Server- und Clientanwendung je ein Thread für das Warten verbraucht wurde. Threads sind aber recht teure Objekte des Betriebssystems, mit denen man sparsam haushalten sollte.

Mit der Servlet Spezifikation 3.0, die im Tomcat 7.0 umgesetzt ist, steht nun die Möglichkeit der asynchronen Verarbeitung von Anfragen zur Verfügung. Sie erlaubt es, die Kommunikations-Streams im Server noch offen zu halten, nachdem die Servicemethode des Servlets zurückgekehrt ist. So kann zu einem quasi beliebigen Zeitpunkt aus einem beliebigen Thread heraus eine Antwort an den Client gesendet werden.

Auch auf der Clientseite stellt sich das Problem des wartenden Threads. Sofern die Anwendung aber nicht selbst eine Serveranwendung ist, die viele Verbindungen zum Indexserver aufmacht (z. B. Web-Client), ist der zusätzliche Thread nicht tragisch.

4 Indexserver JSON API

Ab der Version 8.00.014 können Indexserver Anwendungen auch in JavaScript für Webbrowser geschrieben werden. Die Kommunikation erfolgt synchron oder asynchron mithilfe des XMLHttpRequest Objekts, das von allen gängigen Webbrowsern bereitgestellt wird. Die Anfragen und Ergebnisse werden im JSON Format übertragen. Es stehen alle Indexserver Aufrufe zur Verfügung, bis auf lokale Speichern von Dokumenten. Das Hochladen von Dokumenten ist mit freien JavaScript-Bibliotheken, die auf dem HTML-Input-File-Element basieren, möglich.

4.1 Einführungsbeispiel: Anzeige eines Dokuments nach Objekt-ID

Das folgende Beispiel zeigt die Anmeldung am Indexserver, das Lesen der Daten eines Objekts und die Anzeige der Dokumentendatei in einem zweiten Browserfenster oder –Tab.

Es besteht aus einer HTML Datei, die ein einfaches Formular, wie in Abbildung 7 gezeigt, darstellt. Im Eingabefeld wird eine Objekt-ID erwartet, und beim Klick auf den Button „get“ wird das zugehörige Dokument angezeigt.

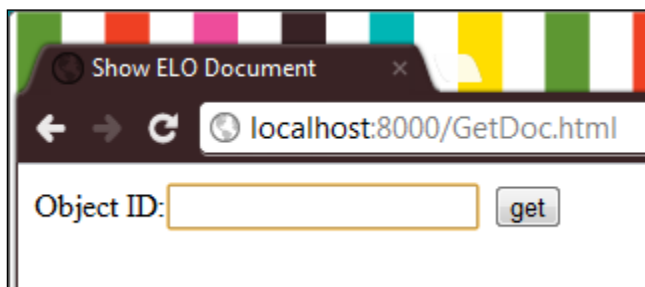


Abb. 5: Formular des Einführungsbeispiels im Browser

Der HTML-Code sieht wie folgt aus.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Show ELO Document</title>

<!-- Include Indexserver JavaScript Library -->
<script type="text/javascript" src="EloixClient.js"></script>

<!-- Example code -->
<script type="text/javascript">

// Simplify namespace access
// (all ELOix symbols are member of de.elo.ix.client)
var IX = de.elo.ix.client;

// Indexserver URL
```

```
// Since cross domain requests are not supported, Indexserver must be
reachable
// via the same server from where this example HTML file is loaded.
var ixUrl = "/ix-elo80/ix";

/**
 * This function reads the object given by its ID in the element
"searchTerm".
 * If the object exists, it navigates the browser to its URL.
 * If the object does not exist, it opens an alert box and displays the
error message created by Indexserver.
 */
function Example_getDoc() {
    try {
        // Get Indexserver connection
        var connFact = new IX.IXConnFactory(ixUrl, "GetDoc", "1.0");
        var conn = connFact.create("Administrator", "elo");

        // Read API constants
        var CONST = conn.getCONST();

        // Get object ID from HTML element
        var objId = document.getElementById("searchTerm").value;

        // Read object
        var ed = conn.ix().checkoutDoc(objId, "", CONST.EDIT_INFO.mbDocument,
CONST.LOCK.NO);

        // Assigning URL to download link
        document.getElementById("downloadLink").href =
            ed.document.docs[0].url + "&acode=attachment";

        // Assign URL to the link that shows the document in the browser
        document.getElementById("viewLink").href =
            ed.document.docs[0].url; // + "&acode=inline" is default
    }
    catch (ex) {
        // Show message box with error text.
        alert(ex.toString());
    }
    finally {
        // Skip logout in this example.
        // Otherwise the document URL might not be valid when the browser
loads it.
    }
};
</script>
</head>
<body>
    Object ID:<input id="searchTerm" type="text" value=""/>
```

```
<input id="bnSearch" type="button" value="get" onclick="Example_getDoc()"
<a href="about:blank" id="downloadLink">Download</a><br/>
<a href="about:blank" id="viewLink">View</a><br/>
</body>
</html>
```

Die erste interessante Zeile des HTML-Codes ist

```
<script type="text/javascript" src="EloixClient.js"></script>
```

Sie hat die Aufgabe, die JavaScript-Klassen und –Funktionen einzubinden, die für die Ansteuerung des Indexservers benötigt werden. Die Datei kann von der Statusseite des Indexservers heruntergeladen werden. Es gibt sie als ausführliche Version mit Kommentaren (EloixClient.js) und als komprimierte Version ohne Kommentare und Formatierungen (EloixClient-min.js). Letztere ist nur ein Drittel so groß und sollte wegen der kürzeren Ladezeit in produktiven Anwendungen benutzt werden.

Im darauffolgenden Script-Block ist die Programmlogik für das Beispiel definiert. Er startet damit, einen einfacheren Zugriff auf die Indexserver-Klassen bereitzustellen.

```
var IX = de.elo.ix.client;
```

Denn die Indexserver-Symbole sind im Package „de.elo.ix.client“ enthalten und dieses muss jedem Klassennamen vorangestellt werden. Um Tipparbeit im Beispielcode zu sparen, wird zum langen Package-Namen der Alias „IX“ definiert. Statt „de.elo.ix.client.IXConnFactory“ kann somit kürzer „IX.IXConnFactory“ geschrieben werden.

Die Zeile

```
var ixUrl = "/ix-elo80/ix";
```

Legt die URL zum Indexserver fest. Weil die Webbrowser im JavaScript nur erlauben, Anfragen an denjenigen Server zu stellen, von dem auch die HTML-Seite stammt, ist die URL hier relativ zum Server angegeben. Entweder muss also die HTML-Seite vom Server kommen, auf dem auch der Indexserver läuft. Oder bestimmte Maßnahmen müssen dies dem Browser glauben machen. Der Abschnitt 2 beschäftigt sich ausführlich mit diesem Thema.

Beim Klick auf den „get“ Button wird die Funktion „Example_getDoc“ ausgeführt. Sie erstellt im ersten Schritt eine Verbindungs-Factory, von der aus mit der „create“ Funktion eine Anmeldung am Indexserver erfolgt.

```
var connFact = new IX.IXConnFactory(ixUrl, "GetDoc", "1.0");
var conn = connFact.create("Administrator", "elo");
```

Lesern, die bereits mit der Java- oder .NET-Schnittstelle gearbeitet haben, wird dieses Vorgehen bekannt vorkommen. Überhaupt ist die JSON-API sehr eng an die bestehenden Indexserver-APIs angelehnt.

In einem Punkt gibt es aber einen deutlichen Unterschied: die Konstanten sind nicht über statische Klassenelemente zugreifbar. Sie müssen wie bei der SOAP-Schnittstelle aus dem Konstanten-Objekt geholt werden. Im Beispiel wird das Konstantenobjekt für den einfacheren Zugriff in einer Hilfsvariablen gespeichert:

```
var CONST = conn.getCONST();
```

Die darauf folgende Zeile

```
var objId = document.getElementById("searchTerm").value;
```

liest die Objekt-ID aus dem HTML-Eingabefeld.

Sie wird der Funktion „checkoutDoc“ übergeben, die die Dokumentdaten liest.

```
var ed = conn.ix().checkoutDoc(objId, "", CONST.EDIT_INFO.mbDocument,  
CONST.LOCK.NO);
```

Ist das Dokument vorhanden, enthält die Rückgabe von „checkoutDoc“ in „ed.document.docs[0]“ die Daten zur aktuellen Arbeitsversion des Dokuments. Insbesondere findet sich in „ed.document.docs[0].url“ die URL, unter der das Dokument vom Browser geladen werden kann.

Die URL kann direkt einem Anker-Element zugewiesen werden. Wenn der String „acode=attachment“ angehängt wird, dann erhält der Browser die Anweisung die Datei herunterzuladen, statt sie im Browserfenster anzuzeigen.

```
... ed.document.docs[0].url + "&acode=attachment";
```

In diesem Beispiel wird auf eine Abmeldung des Anwenders bewusst verzichtet. Denn üblicherweise lädt der Browser das Dokument in einem parallelen Prozess (oder Thread). So lange muss die Sitzung auf jeden Fall gültig sein.

In der Praxis könnte man die Abmeldung in eine Timer-Funktion packen. Oder man sorgt zumindest dafür, dass nicht bei jedem Klick auf „get“ eine neue Sitzung erstellt wird und speichert sie in einer globalen Variablen.

4.2 Cross-Domain Anfragen

Wer mit dem Browser-Objekt XMLHttpRequest bereits gearbeitet hat, ist bestimmt schon auf die Einschränkung gestoßen, dass es aus Sicherheitsgründen nur HTTP-Anfragen zu demjenigen Server erlaubt, von dem auch die HTML Seite kommt. Dies wird häufig als „Same Origin Policy“ bezeichnet.

Diese Einschränkung ist insbesondere während der Entwicklung einer Anwendung störend. Sie kann aufgehoben werden, wenn die HTML-Datei mit dem HTTP-Header „Access-Control-Allow-Origin : <server>“ ausgeliefert wird. Dies ist allerdings oft nicht einfach möglich, weil die HTTP-Header nicht in der HTML-Datei enthalten sind sondern vom Server vergeben werden. (s. http://en.wikipedia.org/wiki/XMLHttpRequest#Cross-domain_requests)

Programmierer von JavaScript-Bibliotheken, wie z. B. jQuery, umgehen die Sicherheitsprüfung, indem sie in den DOM-Baum des Dokuments ein `<script>` Tag einfügen und dies mit der gewünschten URL laden. Das funktioniert, weil dieses Tag nicht der „Same Origin Policy“ unterliegt und Daten von beliebigen Servern laden kann. Nachteil dieser Lösung ist, dass der Aufruf nicht synchron erfolgen kann. Mit asynchronen Aufrufen wird aber die Programmlogik schnell sehr unübersichtlich.

Die einfachste Lösung für das Cross-Domain-Problem ist der Einsatz eines Reverse-Proxy. Dieser verteilt die Anfragen an beliebig viele Server und stellt sich dem Browser gegenüber als ein einziger Server dar. So kann man seine HTML Seiten auf dem einen Server entwickeln und den Indexserver auf einem anderen Server ansprechen.

Auch in der Praxis ist dies eine günstige Konfiguration, wenn man über das Internet auf ELO-Daten zugreifen will. Der Reverse-Proxy läuft dafür in der DMZ und leitet die Anfragen an die Server im geschützten Intranet weiter.

Im Abschnitt 8 wird gezeigt, wie mit wenigen Schritten ein Reverse-Proxy aufgesetzt werden kann.

4.3 Synchron und asynchrone Aufrufe

Bis auf die Anmeldung über die create-Funktionen der IXConnFactory können alle Funktionen der API auch asynchron aufgerufen werden. Hierzu ist im letzten Parameter ein `de.elo.ix.client.AsyncCallback`-Objekt mit Callback-Funktionen zu übergeben.

Das folgende Beispiel zeigt den asynchronen Aufruf der `findFirstSords`-Funktion. Dem Callback-Objekt werden neben den Rückruffunktionen weitere Elemente hinzugefügt, um später ggf. `findNextSords` aufrufen zu können. Insbesondere werden auch die Icons für die Dokumente bereitgestellt (s. `checkoutSordTypes`), sodass die Funktion „`Example_printSord`“ das zum Dokument gehörende Icon in einem HTML-Image-Element anzeigen kann.


```
/**
 * This function performs a fulltext search asynchronously.
 * The search term is read from the HTML element with id="searchTerm".
 * The result list is printed into the HTML table with id="searchResults".
 */
function Example_doSearchAsync() {

    Example_clearResults();

    try {

        // Get Indexserver connection
        var conn = getConn();

        // Get Indexserver API constants
        var CONST = conn.getCONST();

        // Setup FindInfo
        var findInfo = new IX.FindInfo();
        findInfo.findByFulltext = new IX.FindByFulltext();
        findInfo.findByFulltext.term =
document.getElementById("searchTerm").value;

        // Setup Callback object
        var cb = new IX.AsyncCallback(Example_receivedResult,
                                     Example_receivedException);

        cb.conn = conn;
        cb.idx = 0;
        cb.max = 10;
        cb.sordZ = new IX.SordZ(CONST.SORD.mbMin, CONST.SORD.mbDocVersion);
        cb.sordTypes = conn.ix().checkoutSordTypes(null,
CONST.SORD_TYPE.mbAllJPG,
                                     CONST.LOCK.NO);

        // Start search.
        conn.ix().findFirstSords(findInfo, cb.max, cb.sordZ, cb);
    }
    catch (ex) {
        alert(ex.toString());
    }
    finally {
    }
};

/**
 * This function is called with the results returned by findFirstSords or
 * findNextSords.
 * It prints the results into the HTML table "searchResults" and calls
 * findNextSords,
 * if more results are available.
 * @param findResult de.elo.ix.client.FindResult Object

```

```
*/
function Example_receivedResult(findResult) {

    // "this" refers to the AsyncCallback Object passed to findFirstSords.

    // Write results into HTML table
    for (var i = 0; i < findResult.sords.length; i++) {
        Example_printSord(findResult.sords[i], this.sordTypes);
    }

    // If there are more results...
    if (findResult.moreResults) {

        // Invoke findNextSords asynchronously.
        this.idx += findResult.sords.length;
        findResult = this.conn.ix().findNextSords(findResult.searchId,
this.idx,
this);
    }
    else {

        // No more results: close the find handle.
        this.conn.ix().findClose(findResult.searchId);
    }
}

/**
 * This function is called, if findFirstSords or findNextSords returns an
exception.
 */
function Example_receivedException(ex) {

    // Show message box with error.
    alert(ex.toString());
};

/**
 * This function appends the given sord to the result table.
 * @param sord Sord object
 * @param sordTypes Array of SordType objects
 */
function Example_printSord(sord, sordTypes) {

    // Create a row in the result table
    var tr = document.createElement("TR");
    document.getElementById("searchResults").appendChild(tr);

    // Create the first column
    var td = document.createElement("TD");
    tr.appendChild(td);
}
```

```
// The column content
var item = "";

// Find the icon related to the given sord
for (var i = 0; i < sordTypes.length; i++) {
    var sordType = sordTypes[i];
    if (sordType.id == sord.type) {

        // Create an HTML img element
        // and assign the Base64 encoded JPEG icon data to its src
        element.
            item += "<img src=\"data:image/jpeg;base64,\" +
sordType.icon.data + "\"/>";
        break;
    }
}

// Create an anchor for the document.
// If the anchor is clicked, it shows the document in the iframe next
to the table.
// The browser cannot display all documents this way. If it cannot do
so, it prompts
// for downloading the file.
item += "<a href=\"javascript:Example_showDocument('\" +
sord.docVersion.url +
                                                                    '\")\">\" + sord.name
+ "</a>";

// Set column content
td.innerHTML = item;

// Second column with document size or text "Folder"
td = document.createElement("TD");
tr.appendChild(td);
td.innerHTML = sord.docVersion ? sord.docVersion.size : "Folder";
};

/**
 * This function displays the given URL in the iframe next to the result
table.
 */
function Example_showDocument(url) {
    document.getElementById("searchPreview").src = url;
};
<body>
<table>
    <tr>

        <!-- Search form -->
        <td valign="top">
```

```
<div id="searchForm">
  Search Term:<input id="searchTerm" type="text" value="ELO Update"/>
  <input id="bnSearch" type="button" value="Search"
    onclick="Example_doSearchAsync()" />
</div>

<!-- Result table -->
<div style="overflow: scroll; width: 700px; height: 700px;">
<table border="1">
  <thead>
    <tr>
      <th>Short description</th>
      <th>File Size</th>
    </tr>
  </thead>

  <!-- This element will be filled with search results -->
  <tbody id="searchResults">
  </tbody>
</table>
</div>
</td>

<!-- Preview frame shows the selected document -->
<td valign="top"><iframe id="searchPreview" src="about:blank"
  width="500" height="700" frameborder="1"></iframe></td>
</tr>
</table>
</body>
```

Seit ELO 9 kann statt eines Objekts vom Typ `de.elo.ix.client.AsyncResult` auch ein Funktionsobjekt übergeben werden, das wie folgt definiert ist:

```
function(result, exception) {
  if (exception) {
    // Fehler ausgeben
  }
  else {
    // result verarbeiten;
  }
};
```

4.4 Datei Upload

Mithilfe ausgefeilter JavaScript Programmierung auf der Basis des HTML-File-Input Elements ist es möglich, auch Dateien als Dokumente oder Attachments ins Archiv einzuchecken. Derartige Lösungen gibt es in kommerziellen und freien Bibliotheken und manche setzen die Bibliothek jQuery voraus. Der Indexserver JSON-API ist aktuell keine File-Upload-Funktionalität beigelegt, weil sie bisher hauptsächlich im ELO-Webclient eingesetzt wird und dieser die Upload-Funktionen von ExtJS, einer kommerziellen JavaScript-Bibliothek, benutzen kann.

Eine einfache freie Bibliothek stellt Andrew Valums auf der Webseite <http://valums.com/ajax-upload/> zur Verfügung. Sie setzt keine anderen Bibliotheken voraus und eignet sich deshalb besonders gut für ein Beispiel.

In die demo.htm von Valums File-Upload-Beispiel ist die JSON-Lib des Indexservers im HTML-head einzubinden:

```
<head>
...
<!-- Include Indexserver JavaScript Library -->
<script type="text/javascript" src="/ix-elo80/EloixClient.js"></script>
```

Daran anschließend folgt der Beispielcode mit dem Ereignishandler Example_onComplete, der nach erfolgreichem Upload aufgerufen wird.

```
<!-- Example code -->
<script type="text/javascript">

// Simplify namespace access
// (all ELOix symbols are member of de.elo.ix.client)
var IX = de.elo.ix.client;

// Indexserver URL
var ixUrl = "/ix-elo80/ix";

// Login credentials
var userName = "Administrator";
var userPwd = "elo";

// Indexserver connection
var _conn;

/**
 * Returns the Indexserver connection.
 * Creates a new connection, when called the first time.
 */
function getConn() {
    if (!_conn) {
        var connFact = new IX.IXConnFactory(ixUrl, "SimpleSearch", "1.0");
```

```
        _conn = connFact.create(userName, userPwd);
    }
    return _conn;
};

function Example_onComplete(id, fileName, responseJSON) {
    try {
        var conn = getConn();
        var CONST = conn.getCONST();

        var ed = conn.ix().createDoc("1", "", "",
CONST.EDIT_INFO.mbSordDocAtt);
        ed.sord.name = fileName;

        ed.document.docs = [ new IX.DocVersion() ];
        ed.document.docs[0].ext = fileName;

        // Indexserver has created an element "uploadResult" in the
response text.
        // It has to be sent back in order to attach the uploaded file
with the
        // document version.
        ed.document.docs[0].uploadResult = responseJSON.uploadResult;

        ed.document = conn.ix().checkinDocEnd(ed.sord, CONST.SORD.mbAll,
                                                ed.document, CONST.LOCK.NO);

        return true;
    }
    catch (ex) {
        alert(ex);
        return false;
    }
}

</script>

</head>
```

Die Funktion createUploader der demo.htm ist wie folgt zu erweitern:

```
...
    function createUploader(){

        var conn = getConn();

        // Create an URL used as the destination for the file upload.
        // Append a parameter in order to distinguish different upload
solutions,
        // because each solution expects its own JSON response format.
        var actionUrl = conn.makeUploadUrl("&solution=valums");
```

```
var uploader = new qq.FileUploader({
    element: document.getElementById('file-uploader-demo1'),
    action: actionUrl,
    debug: true,
    onComplete: Example_onComplete
});
...
}
```

Mit dem Aufruf `conn.makeUploadResult` wird eine URL zusammengestellt, zu der die Datei hochgeladen wird. Beim Hochladen wird noch kein Dokument oder Attachment erzeugt. Es wird die Datei lediglich, mit einem eindeutigen Namen versehen, ins Temp-Verzeichnis des Servers gelegt. Der Indexserver gibt eine Art ID auf die Datei zurück: das `uploadResult`. Es ist im obigen Beispielcode im Ereignishandler `Example_onComplete` im Parameterobjekt `responseJSON` enthalten und wird dem `DocVersion`-Objekt vor `checkinDocEnd` zugewiesen. Auf diese Art wird die hochgeladene Datei mit einer Dokumentversion verbunden.

Weil jede File-Upload-Lösung ihr eigenes Rückgabeformat vom Server erwartet, kann die Antwort in einem JavaScript-Ereignis zusammengestellt werden. Dazu muss eine JavaScript-Datei in den Archivordner „Administration\Indexserver Scripting Base_ALL“ (bzw. ... \Servername) eingestellt werden, die eine Funktion namens `onFileUploadBuildResponse` enthält. Sie wird mit einigen Informationen zur Datei in Form eines `DocVersion`-Objekts versorgt (Parameter `dv`). Außerdem werden die wichtigsten Informationen aus der Anfrage im Parameter `requestInfo` bereitgestellt: HTTP-Header, Cookies und URL-Parameter. Die Funktion erwartet, dass die Ergebnisse in den Parameter `responseInfo` eingetragen werden. Es sind der Content-Typ und der Ergebnistext einzutragen. Gewöhnlich ist er eine String-Repräsentation eines Java-Objekts (JSON) und muss das vom Indexserver gebildete `uploadResult` enthalten.

```
importPackage(Packages.de.elo.ix.client);
importPackage(Packages.java.util);

/**
 * Indexserver calls this event after a file has been uploaded via the
 * JSON-API.
 * https://github.com/blueimp/jQuery-File-Upload/wiki
 * https://github.com/blueimp/jQuery-File-Upload/wiki/Setup
 * @param ec Execution context
 * @param dv DocVersion object with the following valid members:
 * ext, size, guid, uploadResult, physPath.
 * DocVersion.uploadResult has to be returned to the client application.
 * It must be set into its DocVersion.uploadResult before checkinDocEnd can
 * be called.
 * DocVersion.physPath is set to the location of the uploaded file in the
 * servers temporary directory.
 * @param fileName Name of the uploaded file.
```

```
* @param requestInfo Information of the underlying HttpServletRequest
object.
* @param responseInfo Information for the underlying HttpServletResponse
object.
*/
function onFileUploadBuildResponse(ec, dv, fileName, requestInfo,
responseInfo) {
    log.info("onFileUploadBuildResponse(");
    log.info("dv=" + dv);
    log.info("dv.ext=" + dv.ext);
    log.info("dv.size=" + dv.size);
    log.info("dv.physPath=" + dv.physPath);

    log.info("fileName=" + fileName);

    log.info("requestInfo=" + requestInfo);
    log.info("requestInfo.headers=" + Arrays.toString(requestInfo.headers));
    log.info("requestInfo.cookies=" + Arrays.toString(requestInfo.cookies));
    log.info("requestInfo.requestParams=" +
Arrays.toString(requestInfo.requestParams));

    log.info("responseInfo=" + responseInfo);

    // Find request parameter named "solution".
    // The function conn.makeUploadUrl("&solution=valums") called in the
browsers
    // JavaScript code has appended this parameter to the action URL.
    var solution = getValueFromKey(requestInfo.requestParams, "solution",
"valums");

    if (solution != "valums") {
        throw new "Unknown File-Upload solution: " + solution;
    }

    // Set the Content-Type header of the response.
    responseInfo.contentType = "text/html";

    // Assign the response text: build a JavaScript object with
// members "uploadResult" and "success" and convert it into
// a JSON string.
    var response = { uploadResult : "+dv.uploadResult, success : true };
    responseInfo.responseString = JSON.stringify( response );

    log.info("responseInfo=" + responseInfo);
    log.info(")onFileUploadBuildResponse");
}

// Helper function to find a value by a given key in an array of KeyValue
objects.
function getValueFromKey(keyValues, key, defaultValue) {
    var value = defaultValue;
```



```
for (var i = 0; i < keyValues.length; i++) {  
    if (keyValues[i].key == key) {  
        value = keyValues[i].value;  
        break;  
    }  
}  
return value;  
}
```

4.5 Anmeldung

Zur Anmeldung stehen drei Verfahren zur Auswahl.

4.5.1 Anmeldung mit Anwendername und -kennwort

```
var conn = de.elo.ix.client.IXConnFactory.create(userName, userPassword);
```

Das Kennwort wird hierbei RSA-verschlüsselt versendet.

4.5.2 SSO-Anmeldung: IXConnFactory.createSSO

```
var conn = de.elo.ix.client.IXConnFactory.createSSO();
```

Der Browser übernimmt die Authentifizierung über HTTP-Header zu dem Server, auf dem der Indexserver läuft. Die Prüfung der Anwenderdaten erfolgt unter Windows gegen das ActiveDirectory. Wenn der Benutzer sich dort anmelden kann, erstellt der Indexserver für ihn eine ELO-Sitzung. Es kann auch ein vorgeschalteter Apache2 oder IIS (Reverse-Proxy) die Authentifizierung übernehmen. Dann muss der Tomcat, auf dem der Indexserver läuft, mit dem Modul „mod_jk“ angebunden werden.

4.5.3 Übernahme eines bestehenden Tickets

```
var conn = de.elo.ix.client.IXConnFactory.createFromTicket(ticket);
```

Das Ticket kann z. B. von einer Webanwendung in die dynamisch generierte HTML-Seite oder in den JavaScript-Code eingetragen werden.

4.6 Elementselektoren

Elementselektoren geben an, welche Datenelemente einer API-Klasse bei einem Funktionsaufruf gültig sind. In `checkinSord` gibt der Parameter `sordZ` beispielsweise an, welche Datenelemente des übergebenen Sord-Objekts geschrieben werden sollen.

In der Java- und .NET-API sind die Elementselektoren 64bit-Integer-Werte. Jedes Bit steht für ein bestimmtes Datenelement. JavaScript-Integer sind aber auf höchstens 32bit beschränkt. Aus diesem Grund wurde eine andere Darstellung in der JSON API gewählt: Elementselektoren werden durch Strings repräsentiert. Sie enthalten eine oder eine Kette von Dezimaldarstellungen von 64bit Werten. Jeder Wert endet dabei mit einem Punkt.

Beispiel: Die Indexwerte eines Dokuments speichert das Element „objKeys“ der Klasse „Sord“. Der zugehörige Elementselektor heißt „mbObjKeys“ und befindet sich in der Konstantenklasse „SordC“. In der Java- und .NET-API ist der Elementselektor definiert als 64bit Integer, in dem das Bit 43 gesetzt ist:

```
public final static long mbObjKeys = (1L << 43);
```

Wenn der Elementselektor über die JSON API abgefragt wird (CONST.SORD.mbObjKeys), erhält man den String „8796093022208.“.

Eine Kombination von Elementselektoren kann durch String-Addition (Verketteten) erfolgen.

Beispiel: Zusammenstellen eines Elementselektors für das Lesen der Kurzbezeichnung und der Indexwerte eines Objekts.

```
var conn = ... // IXConnFactory.create...
var objId = ...
var CONST = conn.getCONST();

// Create element selector.
// Constructor computes the bitwise OR operation of the given arguments.
var sordZ = new de.elo.ix.client.SordZ(CONST.SORD.mbName +
CONST.SORD.mbObjKeys);

// Call checkoutSord to read the data.
var sord = conn.ix().checkoutSord(objId, sordZ, CONST.LOCK.NO);
```

Die Verkettung der beiden Elementselektoren hat den Wert „32.8796093022208.“.

Die Funktion checkoutSord kann sowohl mit einem Elementselektor vom Typ SordZ als auch mit EditInfoZ aufgerufen werden. Je nach dem gibt sie ein Sord oder ein EditInfo zurück. Auch für die Funktion createSord gibt es eine solche Sonderbehandlung.

Für checkoutDoc oder createDoc gibt es keine Sonderbehandlung und es muss ein EditInfoZ übergeben werden. Dieser Elementselektor nimmt eine Sonderstellung ein, weil er nicht nur ein Bitset sondern auch ein SordZ enthält. Das nächste Beispiel zeigt, wie man einen EditInfoZ zusammenstellt, der die Kurzbezeichnung des Dokuments, die Haftnotizen und die Maskendefinition selektiert.

```
var conn = ... // IXConnFactory.create...
var objId = ...
var CONST = conn.getCONST();

// Create element selector for EditInfo object
var editZ = new de.elo.ix.client.EditInfoZ(CONST.EDIT_INFO.mbNotes +
CONST.EDIT_INFO.mbMask);
editZ.sordZ = new de.elo.ix.client.SordZ(CONST.SORD.mbName);

// Read the data.
var editInfo = conn.ix().checkoutDoc(objId, "", editZ, CONST.LOCK.NO);
```



Hinweis: Die interne Darstellung der Elementselektoren wurde von ELO 2011 auf ELO 9 geändert. In ELO 2011 besteht ein Elementselektor aus einem Array von Integer-Werten. Daher müssen in den JSON API Programmen alle Stellen geändert werden, wo Elementselektoren kombiniert werden.

4.7 Anwendungsentwicklung, Debugging

Für die Entwicklung von Indexserver Anwendungen mit der JSON API eignen sich sowohl Java-Entwicklungsumgebungen wie bspw. „Eclipse IDE for Java EE Developers „ als auch das „Visual Studio“ von Microsoft.

Das Debugging kann mit den Werkzeugen erfolgen, die in den Webbrowsern integriert sind. Sowohl Chrome, Internet Explorer als auch Firefox bieten komfortable Debugger an. Visual Studio kann im Zusammenspiel mit dem Internet Explorer die Skripte auch in der Entwicklungsumgebung debuggen.

Den Debugger im Chrome findet man unter dem Schraubenschlüssel-Icon über „Tools – Entwicklertools“. Eine Anleitung für den Debugger findet man hier: http://code.google.com/chrome/extensions/tut_debugging.html

Insbesondere während der Entwicklung ist das Cross-Domain-Problem (s. Abschnitt 2) von Bedeutung. Es ist also neben der Entwicklungsumgebung auch ein Reverse-Proxy nötig, der wie in Abschnitt 8 aufgesetzt werden kann.

4.7.1 Entwicklung mit Eclipse

Für die Entwicklung mit Eclipse unter Windows wird benötigt:

- Aktuelle Java Runtime: www.java.com
- „Eclipse IDE for Java EE Developers“: <http://www.eclipse.org/downloads/>
Entpacken in C:\Programme
(Beim ersten Start nach der Wahl des Workspace auf "Workbench" klicken, um in die Entwicklungsumgebung zu gelangen.)
- Tomcat 7.0: <http://tomcat.apache.org/download-70.cgi>, "Binary Distribution", "x-bit Windows zip", nicht "Windows Service Installer"
Entpacken in C:\Programme
- Apache2 Web-Server: <http://httpd.apache.org/download.cgi>
"Win32 Binary without crypto (no mod_ssl) (MSI Installer)"
(Installation erfolgt später wie in Abschnitt 8 beschrieben)

Der Tomcat wird im Eclipse wie folgt eingerichtet:

- Window – Preferences – Server – Runtime Environments – Add...
- Apache Tomcat v7.0
- Installationsverzeichnis des Tomcat eingeben

- Finish

Tomcat läuft standardmäßig auf Port 8080. Wenn er geändert werden muss, dann kann dies geschehen über:

- Ansicht „Servers“ öffnen: Window – Show View – Other ... - Server – Servers
- Doppelklick auf „Apache Tomcat 7“ in der Ansicht „Servers“
- Im Abschnitt Ports die Portnummer(n) anpassen
- Speichern: „File – Save“

Als Entwicklungsprojekt eignet sich ein „Dynamic Web Project“ besonders gut, weil sich die damit erstellte Anwendung in eine WAR-Datei exportieren lässt, die man einfach auf einem beliebigen, anderen Tomcat einspielen kann.

Ein solches Projekt erstellt man über „File – New ... - Project ... - Web – Dynamic Web Project“. Der Projektname muss gewählt werden. Für weitere Eingaben können die Standardwerte akzeptiert werden.

Im neuen Projekt befindet sich der Ordner „WebContent“. In diesem Ordner und seinen Unterordnern können die HTML-Dateien und JavaScripte angelegt werden. Hier sollte als erstes eine Datei namens „index.html“ erstellt werden, die standardmäßig als Einstiegspunkt für die Anwendung dient.

Zum Ausführen der Anwendung öffnet man das Kontextmenü auf dem Projekt und startet „Run As... - Run on Server“. Daraufhin erscheint ein Auswahlfenster, indem man den Server angegeben kann, auf den man die Anwendung deployen möchte. Wir haben oben nur einen Tomcat-Server installiert und haken die Checkbox „Always use this server...“ an, damit der Dialog nicht wieder kommt. Mit „Finish“ startet der Server, und im Eclipse wird die Seite index.html angezeigt.

Eclipse verwendet für die Anzeige von Web-Seiten den Internet Explorer als Plugin. Die angesteuerte URL verweist auf den oben eingerichteten Tomcat. Wegen des Cross-Domain-Problems wird im JavaScript aber keine Verbindung zum Indexserver möglich sein.

Hierfür muss der Apache2 noch installiert und als Reverse Proxy konfiguriert werden, wie es in Abschnitt 8 beschrieben ist.

Anschließend ruft man die Web-Seite unter der Adresse des Apache2 im Browser auf.

4.8 Reverse-Proxy mit dem Apache2 Webserver

Nur wenige Handgriffe sind nötig, um den Apache2 Webserver als Reverse-Proxy zu konfigurieren, damit wir in der Lage sind, Cross-Domain-Anfragen auszuführen. Sicher ist auch der IIS dafür geeignet. Die Konfiguration des Apache2 macht aber einen einfacheren Eindruck.

Auf der Seite <http://httpd.apache.org/download.cgi> kann man den aktuellen Apache2 als Anwendung (Binary) herunterladen, beispielsweise das MSI-Paket für Windows.



Hinweis: Bei der Installation erscheint zweimal eine Eingabeaufforderung, die nicht automatisch verschwindet, wenn der Port 80 bereits von einer anderen Anwendung belegt wird. Die Fenster können dann mit der ESC-Taste geschlossen werden. Unabhängig davon gelingt die Installation. Man muss nur vor dem ersten Start in der Konfigurationsdatei die Portnummer ändern („Listen <port>“) oder den Port frei machen.

Die Konfiguration des Webserver erfolgt in der Datei <instldir>/conf/httpd.conf. Es ist eine einfache Textdatei, die mit dem Notepad bearbeitet werden kann. Zeilen, die mit einem „#“ beginnen, sind Kommentare.

Als erstes muss der Apache2 angewiesen werden, die Module „proxy_module“ und „proxy_http_module“ zu laden. Dazu sind die entsprechenden Zeilen „#LoadModule ...“ zu aktivieren, d.h. das Kommentarzeichen „#“ muss am Zeilenanfang entfernt werden.

```
...
LoadModule proxy_module modules/mod_proxy.so
...
LoadModule proxy_http_module modules/mod_proxy_http.so
...
```

Nun müssen die Module noch parametrisiert werden. Am Ende der Konfigurationsdatei erfolgt dies mit den folgenden Zeilen, die man noch an die konkreten Gegebenheiten anpassen muss.

```
ProxyRequests Off

<Proxy *>
Order deny,allow
Allow from all
</Proxy>
ProxyPass /ix-elo80 http://srvt02:8084/ix-elo80
ProxyPassReverse /ix-elo80 http://srvt02:8084/ix-elo80
```

Die Optionen „ProxyPass“ und „ProxyPassReverse“ haben jeweils gleiche Parameter. Der erste legt fest, welche URI umgeleitet werden soll. Mit dem zweiten wird angegeben, welcher Server den Aufruf bekommt.

Beispiel: Angenommen, der Apache2 wird mit <http://myserver:8000> angesprochen. Dann werden mit obiger Konfiguration Anfragen beginnend mit „/ix-elo80“ an den Server „srvt02“ weitergeleitet. Z. B. wird die Anfrage <http://myserver:8000/ix-elo80/ix> weitergeleitet an <http://srvt02:8084/ix-elo80/ix>. Enthält der Request nicht „/ix-elo80“, erfolgt keine Weiterleitung und der Apache2 sucht den Inhalt in seinem Verzeichnis „<instldir>/htdocs“. Hier könnte beispielsweise die HTML-Datei mit dem JavaScript-Programm liegen.



Hinweis: Wenn der Apache2 zu einem Apache Tomcat weiterleiten soll, kann dies auch über das Modul „mod_jk“ geschehen. Hierbei kommt ein spezielles und optimiertes Protokoll zum Einsatz. Ferner bietet es auch die Möglichkeit, mehrere Tomcats in einer Lastverteilung zusammenzuschalten. Die Konfiguration ist nur geringfügig aufwendiger und wird in vielen Anleitungen im Internet beschrieben.

4.9 Beispiele

In der Indexserver_Programming.zip befinden sich einige Beispiele zur Verwendung der JSON-API im Ordner /doc/JSON API/. Zum Ausprobieren auf einem Tomcat, kopiert man das komplette Verzeichnis in ein Unterverzeichnis unter webapps. Das Unterverzeichnis sollte der Einfachheit halber kein Leerzeichen enthalten, bspw. „JSONAPI“. Die Beispielseite GetDoc.html kann dann aufgerufen werden über: <http://server:port/JSONAPI/GetDoc.html>.

In den Beispielen ist die Indexserver URL fest verdrahtet in der Variablen „ixUrl“ und muss an die Umgebung angepasst werden. Ebenso sind Anwendername und Kennwort vorgegeben. Die Variablen hierzu lauten „userName“ und „userPwd“.

5 ELOix Lastverteilung

Dieses Kapitel beschreibt den Betrieb mehrerer Indexserver für dasselbe Archiv und die Lastverteilung mittels Apache2 Webserver.

5.1 Überblick

In ELOenterprise-Installationen mit großen Anwenderzahlen kann es sinnvoll sein, mehrere Indexserver auf verschiedene Server verteilt zu betreiben. Damit für alle Client-Anwendungen dieselbe Indexserver-URL gilt, schaltet man einen Proxy-Dienst zwischen Client und Server, der die Anfragen gleichmäßig verteilt.

Der Apache2 eignet sich besonders gut für diese Aufgabe wegen seiner einfachen Handhabung und der Möglichkeit, die Kommunikation zu den Servern über ein optimiertes Protokoll abzuhandeln.

5.2 Zusätzlichen Indexserver installieren

Voraussetzung ist eine Java 7.0 Installation und ein Tomcat 7.0.30 oder neuer.

Ein neuer Indexserver kann zwar auch im laufenden Betrieb ergänzt werden. Unter hoher Last kann aber das nötige „Reload“ zu Out-Of-Memory-Fehlern in den Anwendungen führen. Deshalb sollte ein Zeitraum gewählt werden, in dem möglichst wenige Indexserver-Anfragen zu erwarten sind.


	Benennung: das Verzeichnis der ELOenterprise Installation wird im Folgenden instdir1 genannt.	ELOenterprise Verzeichnis: _____
	Benennung: das Verzeichnis der Tomcat-Installation für die ELOenterprise Installation wird im Folgenden tomdir1 genannt.	tomdir1 Verzeichnis: _____
	Benennung: der Indexserver der ELOenterprise Installation benötigt einen (unter allen Indexservern des Archivs) eindeutigen Namen. Wenn nicht explizit ein Name angegeben wurde, ist es der Computernamen.	ixid1: _____
	Benennung: das Verzeichnis der Tomcat-Installation auf dem Zielsystem wird im Folgenden tomdir2 genannt.	tomdir2 Verzeichnis: _____
	Tomcat auf dem Zielsystem stoppen	

	Neues Verzeichnis für die benötigten Dateien auf dem Zielsever erstellen. Z. B. „C:\ELOenterprise“. Es wird im Folgenden instdir2 genannt.	Zielverzeichnis: _____
	Verzeichnis instdir2\webapps anlegen	
	Verzeichnis instdir2\config\ix-Archiv anlegen	
	Verzeichnis instdir2\data\ix-Archiv anlegen	
	Verzeichnis instdir2\logs anlegen	
	Download der Indexserver.zip aus dem Supportweb.	
	Datei elox.war aus der Indexserver.zip in das Verzeichnis instdir2\webapps kopieren	
	Alle Dateien aus dem Verzeichnis der ELOenterprise Installation instdir1\config\ix-Archiv kopieren nach: instdir2\config\ix-Archiv	
	Datei instdir2\config\ix-Archiv\config.xml editieren: Neuen entry-Eintrag hinzufügen: <entry key="ixid">ixid2</entry> entry-Eintrag „ jdbcurl “ kontrollieren (localhost wird i.a. nicht funktionieren) „ixid2“ ist ein eindeutiger Name unter allen Indexserver-Instanzen des Archivs.	ixid2: _____
	Datei instdir2\config\ix-Archiv\log4j.properties editieren: log4j.appender.FI.File=instdir2\logs\ix-Archiv.log	
	Kopieren (einer) der folgenden Dateien (JDBC-Treiber) aus	

	dem Verzeichnis tomdir1\lib ins Verzeichnis tomdir2\lib sqljdbc4.jar (für Microsoft SQL Server) ojdbc6.jar (für ORACLE) db2jcc.jar (für DB2)	
	Connector-Einstellungen der server.xml auf den neuen Tomcat übertragen.	
	Kopieren der Anwenderdefinitionen für die Tomcat-Administration: Kopiere tomdir1\conf\tomcat-users.xml nach tomdir2\conf\tomcat-users.xml	
	Erstellen der XML-Datei tomdir2\conf\Catalina\localhost\ix-Archiv.xml mit dem folgenden Inhalt: <pre><?xml version='1.0' encoding='UTF-8'?> <Context path="/ix-Archiv" docBase="instdir1\webapps\eloix.war" unpackWAR="false"> <Environment name="webappconfigdir" value="instdir1\config\ix-Archiv" type="java.lang.String" override="false"/> </Context></pre>	
	Tomcat auf dem Zielsever noch nicht starten.	

5.2.1 Konfiguration


	Statusseite des Indexservers der ELOenterprise-Installation aufrufen (nicht die Statusseite des gerade installierten)	
	Link „Configure Options“ klicken und mit dem Tomcat Konto anmelden.	

	<p>In der letzten Zeile der Optionentabelle nacheinander die folgenden Einträge hinzufügen (Klick auf „Add“):</p> <table border="1"> <thead> <tr> <th>Instance</th><th>Name</th><th>Value</th></tr> </thead> <tbody> <tr> <td>ixid2</td><td>configDir</td><td>instdir2\data\ix-Archiv</td></tr> <tr> <td>ixid2</td><td>privateUrlBase</td><td>http://tomcat2:port/ix-Archiv/ix</td></tr> </tbody> </table> <p>Wenn für den Indexserver der ELOenterprise Installation (auf tomcat1 mit ixid1) noch keine privateUrlBase angegeben wurde, dann muss dies jetzt geschehen:</p> <table border="1"> <thead> <tr> <th>Instance</th><th>Name</th><th>Value</th></tr> </thead> <tbody> <tr> <td>ixid1</td><td>privateUrlBase</td><td>http://tomcat1:port/ix-Archiv/ix</td></tr> </tbody> </table> <div style="border: 1px solid black; padding: 10px; margin-top: 10px;">  <p>Hinweis: Über die privateUrlBase nehmen die Indexserver gegenseitig Kontakt auf und gleichen Ihre Cache-Daten ab. Die Einstellung darf also nicht auf den Load-Balancer verweisen.</p> </div>	Instance	Name	Value	ixid2	configDir	instdir2\data\ix-Archiv	ixid2	privateUrlBase	http://tomcat2:port/ix-Archiv/ix	Instance	Name	Value	ixid1	privateUrlBase	http://tomcat1:port/ix-Archiv/ix	
Instance	Name	Value															
ixid2	configDir	instdir2\data\ix-Archiv															
ixid2	privateUrlBase	http://tomcat2:port/ix-Archiv/ix															
Instance	Name	Value															
ixid1	privateUrlBase	http://tomcat1:port/ix-Archiv/ix															
	Tomcat auf dem Zielsystem starten.																
	Alle Indexserver des Archivs jeweils auf ihrer Seite „Configure Options“ mit dem Button „Reload“ neu starten (Offene Sitzungen gehen nicht verloren).																

5.3 Apache2 als Lastverteilung


Während der Installation und Konfiguration wird es nötig sein, den ELOenterprise Server zu stoppen.

5.3.1 Apache2 installieren

	Laden Sie dazu den Apache Webserver unter folgender Adresse herunter: http://httpd.apache.org/download.cgi	
	Installieren Sie den Server. Das Zielverzeichnis wird im Folgenden apachedir genannt.	apachedir: _____
	<p>Laden Sie den Apache Tomcat Connector (mod_jk) unter folgender Adresse herunter: http://tomcat.apache.org/download-connectors.cgi</p> <div style="border: 1px solid black; padding: 10px; margin: 10px 0;">  Hinweis: Nutzen Sie den Link Binary Releases auf der Downloadseite, um die zum Apache2 passende Versionen zu finden: z.B. „/windows/tomcat-connectors....-httpd-2.4.x.zip“ </div>	
	Kopieren Sie die „ mod_jk.so “ aus der „tomcat-connectors...zip“ in das Verzeichnis apachedir\modules .	


5.3.2 Apache2 konfigurieren

	<p>Bearbeiten Sie die Datei httpd.conf im Verzeichnis apachedir\conf des Apache2.</p> <p>Ergänzen Sie am Ende:</p> <p>Include conf/tomcat_jk.conf</p>	
	<p>Erstellen Sie die Datei tomcat_jk.conf mit folgendem Inhalt:</p> <pre>#load the JK modul LoadModule jk_module modules/mod_jk.so # include Tomcat-Workers file JkWorkersFile conf/workers.properties # Logfile, level und format JkLogFile logs/mod_jk.log # Log level [debug/error/info] JkLogLevel info JkLogStampFormat "[%a %b %d %H:%M:%S %Y] " # jk shared memory location JkShmFile logs/mod_jk.shm # redirect /webapp to the tomcat loadbalancer JkMount /ix-Archiv loadbalancer JkMount /ix-Archiv/* loadbalancer</pre>	
	<p>Erstellen Sie die Datei workers.properties im Verzeichnis apachedir\conf mit folgendem Inhalt:</p>	

<p>worker.list=loadbalancer</p> <p>worker.tomcat1.port=8009</p> <p>worker.tomcat1.host=server1</p> <p>worker.tomcat1.type=ajp13</p> <p>worker.tomcat1.lbfactor=1</p> <p>worker.tomcat2.port=8009</p> <p>worker.tomcat2.host=server2</p> <p>worker.tomcat2.type=ajp13</p> <p>worker.tomcat2.lbfactor=1</p> <p>worker.loadbalancer.type=lb</p> <p>worker.loadbalancer.balance_workers=tomcat1,tomcat2</p> <p>worker.loadbalancer.method=Busyness</p> <p>worker.loadbalancer.sticky_session=true</p> <div data-bbox="223 1438 1326 1928">  <p>Hinweis: Die Portnummer für worker.*.port entnehmen Sie der server.xml des entsprechenden Tomcat im Connector-Tag, Beispielsweise :</p> <p><Connector port="8009" protocol="AJP/1.3"...></p> <p>Die Bezeichnung tomcat1 (oder tomcat2) benennt einen Tomcat-Server. Sie muss später in der server.xml des Tomcat eingetragen werden, s. 3.3.</p> <p>Für einen weiteren Tomcat-Server ergänzen Sie einen Block worker.tomcat3.* und die Liste bei worker.loadbalancer.balance_workers=tomcat1,tomcat2,tomcat3</p> </div>	
<p>Starten Sie den Apache2 Server neu.</p>	

5.3.3 Tomcat Konfiguration ergänzen

Die Lastverteilung muss dafür sorgen, dass Anfragen eines Clients immer zum selben Server gelangen. Den richtigen Server ermittelt sie aus dem HTTP Cookie JSESSIONID, dass beim ersten Aufruf des Clients vom Server vergeben wird und vom Client bei folgenden Anfragen immer mitgesendet wird. Die Lastverteilung erwartet, dass der Servername des Tomcat (hier also tomcat1 oder tomcat2) an die JSESSIONID angehängt ist. Dazu muss der Tomcat seinen Namen kennen. Die folgenden Schritte beschreiben die hierzu nötige Erweiterung der Tomcat-Konfiguration.

	Editieren Sie die Datei <tomdir1>\conf\server.xml	
	<p>Suchen Sie das XML-Tag <Engine name="Catalina" ...></p> <p>Ergänzen Sie im Tag das Attribut jvmRoute und weisen Sie ihm den Tomcat-Namen tomcat1 zu.</p> <p><Engine name="Catalina" jvmRoute="tomcat1" ...></p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;">  Hinweis: Es gibt möglicherweise mehrere dieser Tags. Achten Sie darauf, dass sie das nicht-auskommentierte verändern. </div>	
	Starten Sie den Tomcat neu.	
	Führen Sie die Schritte auch mit dem zweiten Tomcat in tomdir2 durch. Verwenden Sie als Tomcat-Namen tomcat2 .	

5.3.4 Indexserver Option ixUrlBase setzen

	Statusseite eines Indexservers aufrufen.							
	Link „Configure Options“ klicken und mit dem Tomcat Konto anmelden.							
	<p>In der Optionentabelle die Optionen ixUrlBase wie folgt ergänzen:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 15%;">Instance</th><th style="width: 35%;">Name</th><th style="width: 50%;">Value</th></tr> </thead> <tbody> <tr> <td>ixid1</td><td>ixUrlBase</td><td>http://apache:port/ix-Archiv/ix</td></tr> </tbody> </table>	Instance	Name	Value	ixid1	ixUrlBase	http://apache:port/ix-Archiv/ix	
Instance	Name	Value						
ixid1	ixUrlBase	http://apache:port/ix-Archiv/ix						

	Ixid2	ixUrlBase	http://apache:port/ix-Archiv/ix	
	<p>Hierbei ist apache:port der Servername und Port unter dem der Apache2 extern erreichbar ist. Über diese URL werden die Client-Anwendungen auf Dokumente zugreifen.</p>			

5.3.5 Weitere Informationen

Weitere Informationen zum Apache2 Server und Tomcat Connector finden Sie auf den Web-Seiten der Apache Group.

<http://httpd.apache.org/>

<http://tomcat.apache.org/connectors-doc/index.html>

6 Zugriff auf ein ELO-Archiv über WebDAV

ELO WebDAV (IxWebDav) ist die Implementierung der Standardschnittstelle WebDAV, um die Struktur eines ELO-Archivbaums als Netzlaufwerk darzustellen.

Wie ELO FS beinhaltet ELO WebDAV Funktionen zum Auschecken von Dokumenten, zum Einchecken von neuen Versionen und zur Bearbeitung von Verschlagwortungsinformationen (nur bei einigen WebDAV-Clients möglich). Ergänzend können Ordner und Dokumente gelöscht, kopiert und verschoben werden. ELO WebDAV bietet die Funktion zum Sperren von Dateien und Ordner (wenn ein Ordner gesperrt ist, kann man keine Datei von diesem Ordner löschen und keine neue Dateien in diesem Ordner hinzufügen). Allerdings unterstützen wenige WebDAV-Clients (wie BitKinex, WebDrive, CrossFTP) diese Features.

Im Gegensatz zu ELO FS ist der Zugriff über ELO WebDAV auf das ELO-Archiv komplett von Betriebssystemen unabhängig. WebDAV ist in Windows, Mac OS X und Linux fest in das Betriebssystem integriert. Außerdem können mobile Geräte wie iPhone, iPad, Android-Geräte, Blackberries oder Windows-Mobiltelefone über WebDAV auf ELO-Archive zugreifen.

Ein weiterer Unterschied zu ELO FS besteht darin, dass z.B. Microsoft-Office-Dokumente nicht explizit aus- und wieder eingecheckt werden müssen. Dies geschieht automatisch durch die WebDAV-Client-Implementierung in den Office-Programmen bzw. in Windows.

6.1 ELO WebDAV –Server-Adresse

ELO WebDAV ist bereits in den ELO-Indexserver integriert. Sie erreichen das ELO-WebDAV-Verzeichnis über den Browser. Geben Sie dazu die ELO-WebDAV-Adresse nach folgendem Schema ein:

```
http://<Servername>:<Port>/ix-<Archivname>/ixwebdav
```

6.1.1 Verbindung mit vollqualifiziertem Domännennamen

Wenn Sie den WebDAV-Server über den vollqualifizierten Domännennamen (Fully-Qualified Domain Name, FQDN) verbinden wollen, müssen Sie den entsprechende Server in die Zone *Lokales Intranet* aufnehmen:

Öffnen Sie den Konfigurationsdialog *Internetoptionen*.

Wechseln Sie auf die Registerkarte *Sicherheit*.

Klicken Sie auf die Schaltfläche *Sites*.



Abb. 6: Zone ‚Lokales Intranet‘ bearbeiten

Der Dialog *Lokales Intranet* erscheint.

Die Einstellung *Intranet automatisch ermitteln* muss aktiviert sein.

Klicken Sie auf *Erweitert*.

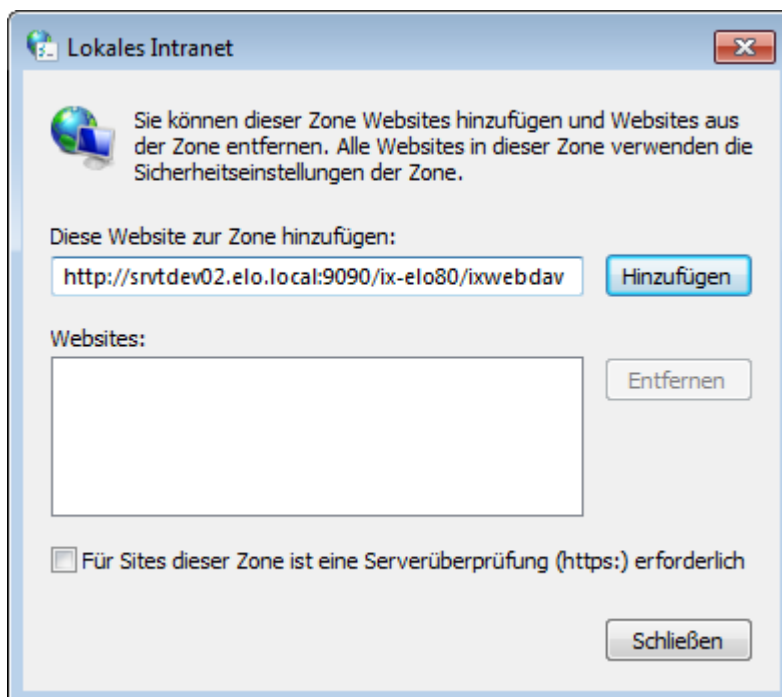


Abb. 7: Vollqualifizierte URL hinzufügen

Ein weiterer Dialog *Lokales Intranet* erscheint.

Tragen Sie die vollqualifizierte URL des Servers ein.

Folgende Beispiele verdeutlichen den Unterschied zwischen einer normalen URL und einer vollqualifizierten URL:

- **Normale URL:** http://srvtdev02:9090/ix-elo80/ixwebdav
- **Vollqualifizierte URL:** http://srvtdev02.elo.local:9090/ix-elo80/ixwebdav

Klicken Sie auf *Hinzufügen*.

Schließen Sie den Dialog.

Schließen Sie die weiteren Dialoge mit *OK*.

6.2 ELO WebDAV unter Windows verwenden

Unter Windows wird die NTLM-Authentifizierung verwendet. D.h. Passwörter werden nur verschlüsselt an den Server übertragen.

6.2.1 Mit Windows Explorer als Netzlaufwerk verbinden (ab Windows Vista)

Um ELO WebDAV als Netzlaufwerk unter Windows (Vista oder höher) mit dem ELO WebDAV zu verbinden, rufen Sie zunächst den Explorer auf und klicken dann mit der rechten Maustaste auf *Computer* oder *Netzwerk*.

Im Kontextmenü klicken Sie auf den Menübefehl *Netzlaufwerk verbinden*.

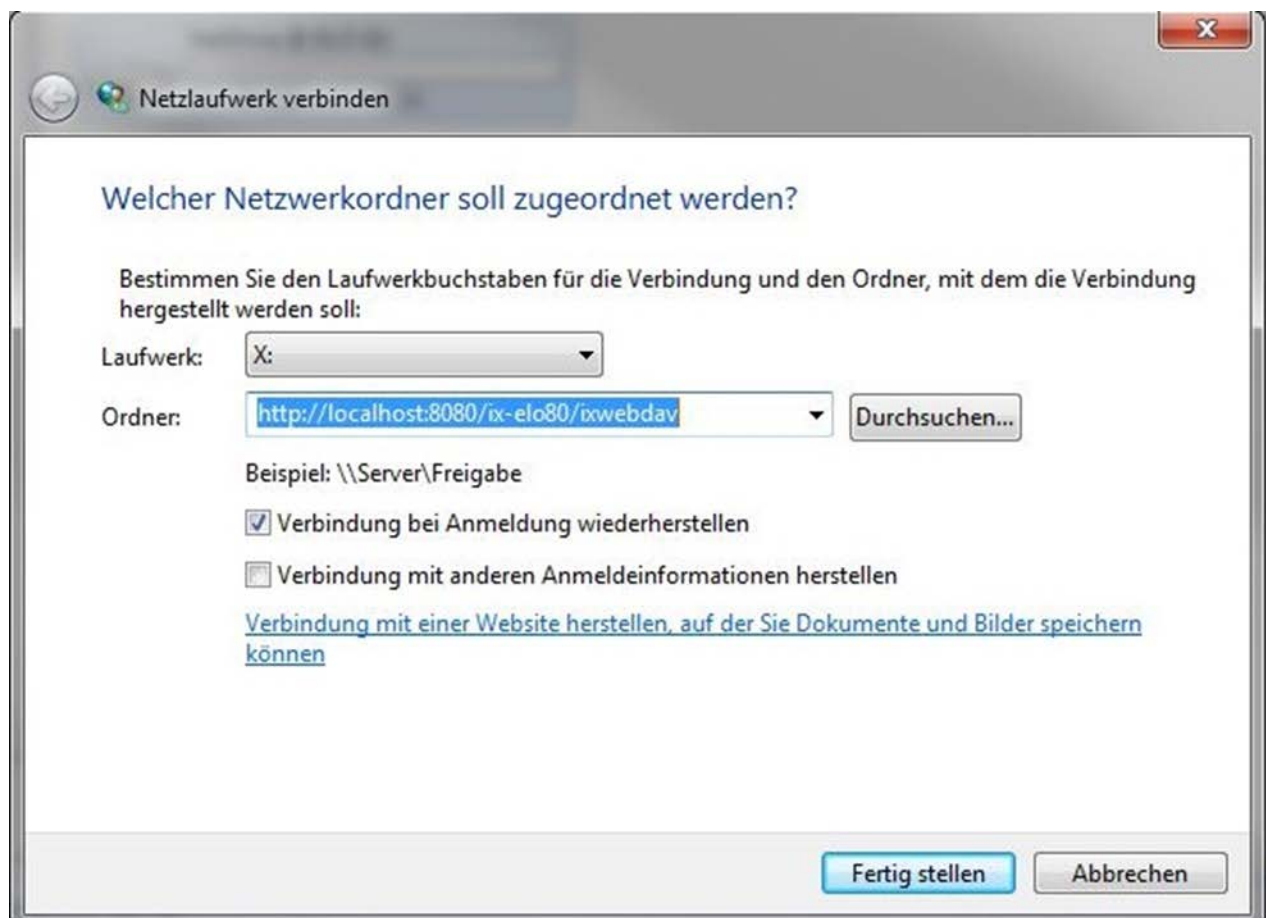


Abb. 8: Netzwerklaufwerk verbinden unter Windows

Wählen Sie einen entsprechend freien Laufwerksbuchstaben und geben unter *Ordner* die Adresse zu Ihren ELO WebDAV ein.

Klicken Sie auf *Fertig stellen*.

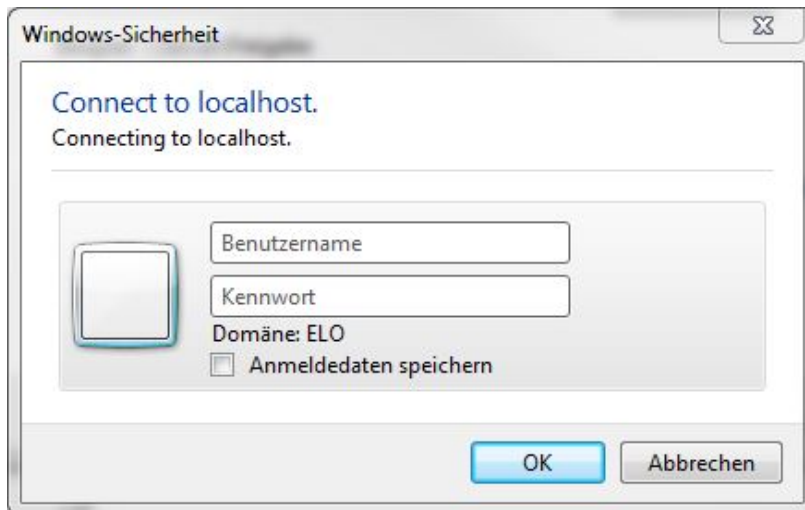


Abb. 9: Login ohne Active Directory

Gehört der angemeldete Windows Benutzer zum dem Active Directory und ist in ELO vorhanden, wird die NTLM-Authentifizierung automatisch angewendet und keine Anmeldeinformationen sind einzugeben. Sonst wird man Dialog Windows-Sicherheit nach Username und Password gefragt. Autorisieren Sie sich mit den ELO-Benutzer-Daten.



Beachten Sie: In diesem Fall wird das Authentifizierungsverfahren BASIC verwendet, und das Password wird als Klartext an den Server übertragen. Sie können die Datenübertragung durch SSL sichern.

6.2.1.1 Langsame Verbindung zum WebDAV-Laufwerk

Wenn die Verbindung zum WebDAV-Laufwerk und Windows Vista oder höher nur mit niedriger Geschwindigkeit erfolgt, beachten Sie folgende Schritte:

Öffnen Sie den Konfigurationsdialog *Internetoptionen*.

Wechseln Sie auf die Registerkarte *Verbindung*.

Klicken Sie auf die Schaltfläche *LAN-Einstellungen*.

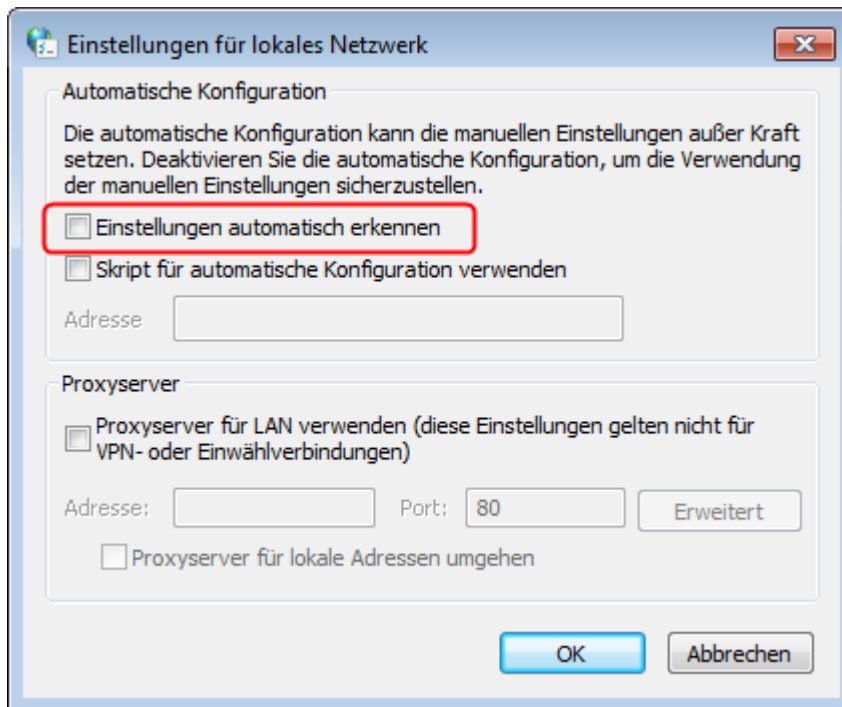


Abb. 10: Option für die lokalen Netzwerkeinstellungen ändern

Der Dialog Einstellungen für lokales Netzwerk erscheint.

Deaktivieren Sie die Option *Einstellungen automatisch erkennen*.

Bestätigen Sie mit *OK*.

Schließen Sie die *Internetoptionen* mit *OK*.

6.2.2 Mit Windows XP

Öffnen Sie den Arbeitsplatz, klicken Sie auf *Extras > Netzlaufwerk verbinden*.

Wählen Sie den gewünschten Laufwerksbuchstaben aus und klicken Sie auf den Link *Onlinespeicherplatz anfordern oder mit einem Netzwerkserver verbinden*.

Der Windows-Assistent zum Hinzufügen von Netzwerkressourcen startet. Klicken Sie auf *Weiter*.

Wählen Sie *Eine andere Netzwerkressource auswählen* aus und klicken Sie auf *Weiter*.

Geben Sie die Adresse des ELO WebDAVs ein und klicken Sie auf *Weiter*.

Geben Sie einen Namen für die Netzwerkadresse ein.

Klicken Sie auf *Fertig stellen*.

6.2.3 Verwendung des WebDAV Redirectors

Unter folgendem Link finden Sie die Informationen über die Verwendung von WebDAV über den WebDAV Redirector (Unter Windows 7, Windows 8, Windows Server 2008 oder Windows Server 2012):

<http://www.iis.net/learn/publish/using-webdav/using-the-webdav-redirector>



Hinweis:

- Für Windows Server ab Version 2008 muss Desktop Experience installiert werden
- Wenn Sie die Fehlermeldung "System Fehler 67 aufgetreten. Der Netzwerkname wurde nicht gefunden" beim Einbinden des ELO-WebDAV als Netzlaufwerk bekommen, bitte prüfen Sie ob der Windows-Dienst WebClient läuft und ob Desktop Experience installiert ist.

6.2.4 Mit der Kommandozeile

Um ELO WebDAV zu einem Netzlaufwerk zu mappen, nutzen Sie in der Windows-Kommandozeile den Befehl:

```
net use Z: \\<Host-Name>:<Port>/ix-<Archivname>/ixwebdav/persistent:yes
```

Über den Parameter `/persistent` können Sie festlegen, ob das Netzlaufwerk auch nach einem Neustart wieder verfügbar sein soll. Setzt man hier den Wert `no`, wird die Verbindung nicht mehr wiederhergestellt.

6.2.5 Mit NetDrive

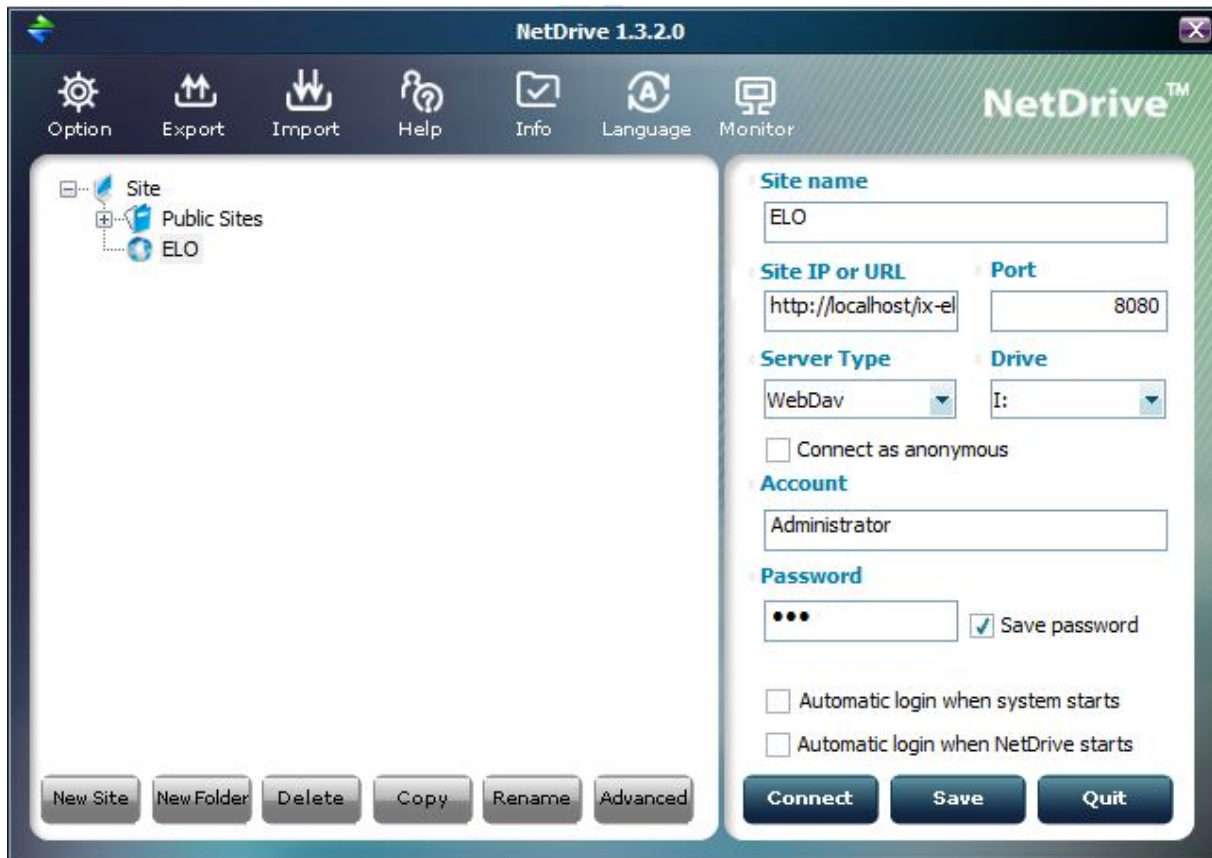


Abb. 11: Zugang über NetDrive

Klicken Sie auf *New Site* und geben Sie einen Namen für die Verbindung ein.

Geben Sie bei *Site IP or URL* die Adresse zum ELO WebDAV ein.

Wählen Sie im Drop-Down-Menü *Server Type* den Eintrag *WebDav* aus.

Tragen Sie in das Feld *Port* den Port des Servers ein

Wählen Sie den Laufwerksbuchstaben aus, den Sie verwenden möchten.

Geben Sie Ihre ELO-Zugangsdaten bei *Account* und *Password* ein.

Um SSL-Verschlüsselung zu aktivieren, klicken Sie auf *Advanced* und aktivieren Sie *Use HTTPS*.

Um SSL-Verschlüsselung zu benutzen, wechseln Sie den Port von **8080** auf **443**

Um die Verbindung zu speichern, klicken Sie auf *Save*.

Um die Verbindung herzustellen, klicken Sie auf *Connect*.

6.2.6 Mit BitKinex

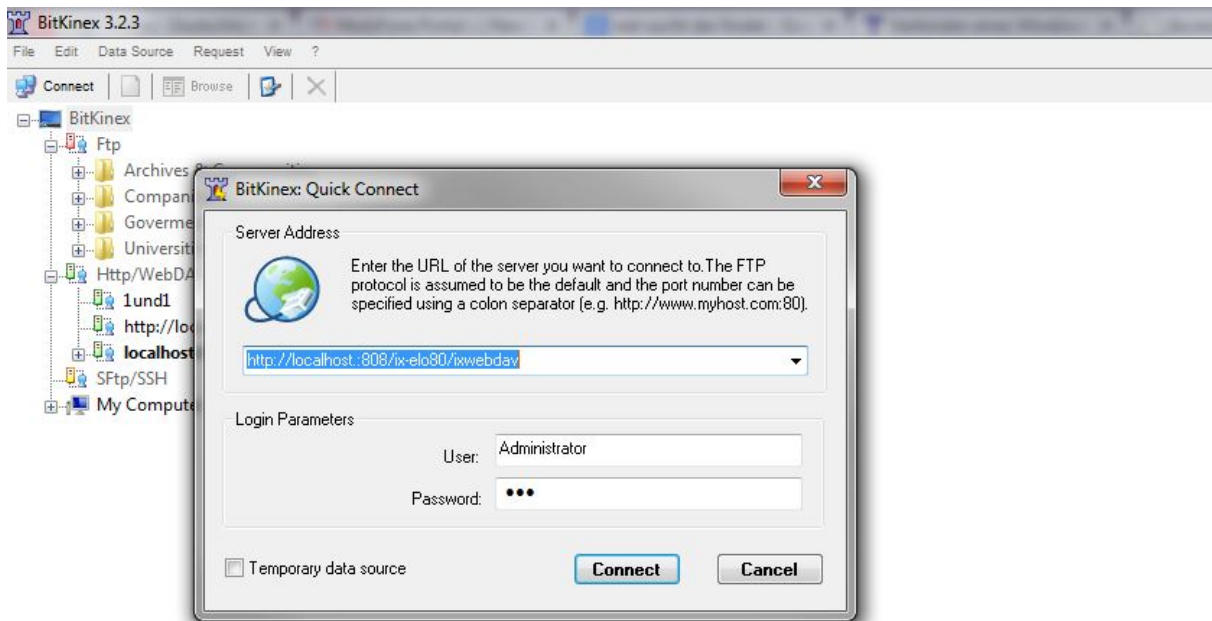


Abb. 12: Zugang über BitKinex

Im Gegensatz zum Windows Explorer und NetDrive unterstützt BitKinex Verbindung mit WebDAV als Netzlaufwerk nicht, sondern verwendet eine eigene Oberfläche. Allerdings bietet BitKinex umfangreiche Features wie das Sperren von Ordner und Dateien, Verschiebung und Kopieren von Dateien im Server ohne Herunterladen von Dateien.

Um BitKinex mit ELO WebDAV zu verbinden, klicken Sie auf *Connect* oder *Quick Connect* im Menü *File*. Geben Sie die Internetadresse zum ELO WebDAV und die ELO-Benutzerdaten ein.

Dann klicken Sie auf *Connect*.

6.2.7 WebDAV unter Microsoft Office Anwendungen verwenden

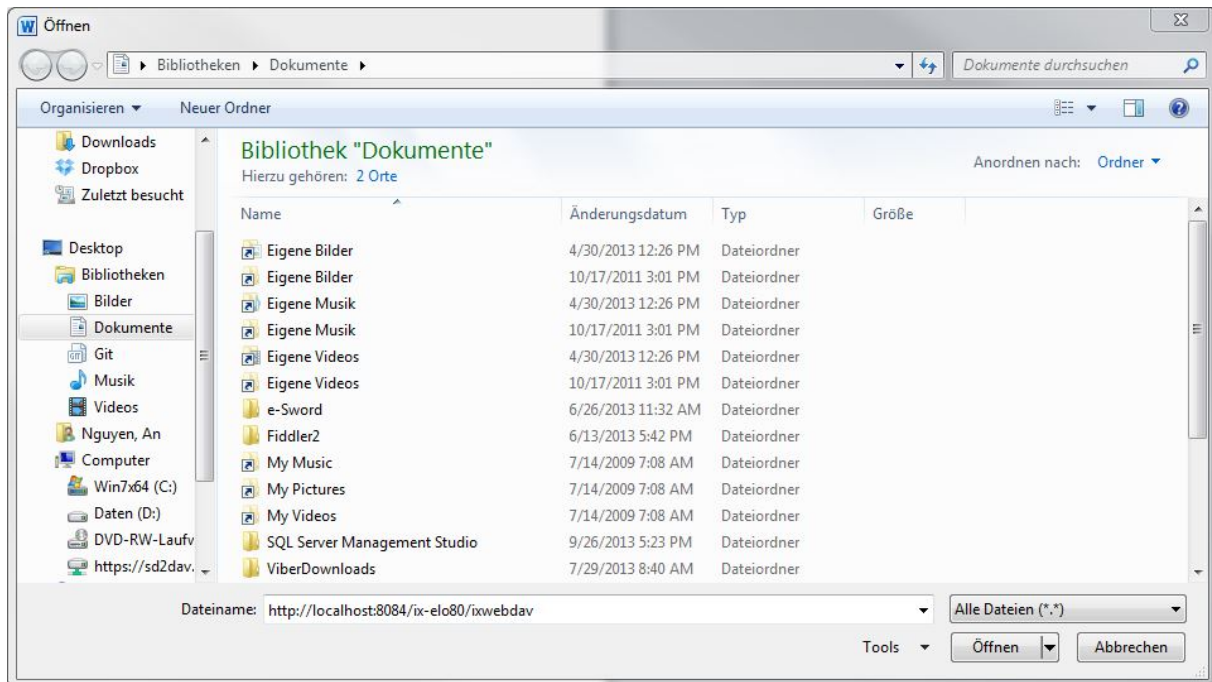


Abb. 13: WebDAV-Anbindung über Microsoft Word

Microsoft Office verwendet einen eigenen WebDAV-Client. Deshalb können Sie ein ELO-Archiv in Office-Anwendungen wie Word, Excel oder PowerPoint öffnen, ohne ELO WebDAV mit einem Netzlaufwerk verbinden zu müssen.

Geben Sie im Dialog *Öffnen* die Internetadresse als Dateiname ein und bestätigen Sie die Eingabe mit der Eingabe-Taste.



Beachten Sie: Klicken Sie nicht auf die Schaltfläche *OK*.



Hinweis: Mit Standardeinstellungen öffnen einige Microsoft-Office-Programme (Microsoft Office 2007 oder früher) Dokumente, die per Hyperlink aufgerufen werden, nur mit Lesezugriff. Um sowohl Lese- als auch Schreibzugriff zu erhalten müssen einige Bedingungen erfüllt sein. Dazu bietet folgende Webseite eine Anleitung: http://www.webdavsystem.com/server/documentation/ms_office_read_only

6.2.7.1 Objekte aus Archiv in MS Office Anwendungen einbetten

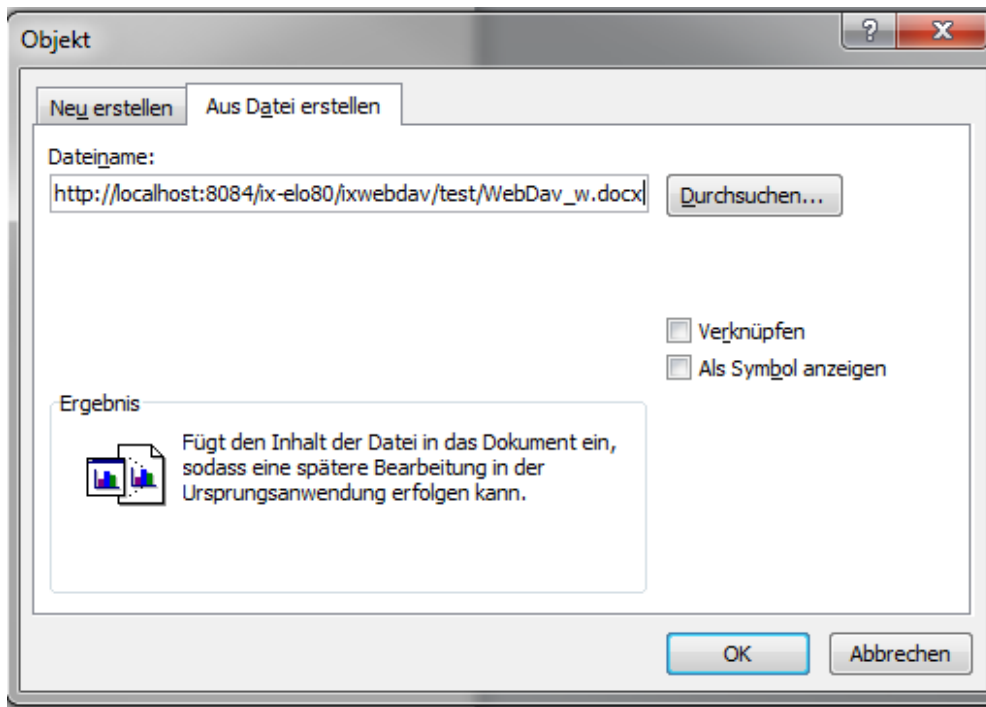


Abb. 14: WebDAV-Objekt einfügen in einer Microsoft-Office-Anwendung

Über ELO WebDAV lassen sich Dateien aus dem ELO-Archiv auch als Objekt einbetten.

6.3 WebDAV unter Linux verwenden

In Linux wird das Authentifizierungsverfahren NTLM nicht verwendet. Stattdessen kommt das Basic-Authentication-Verfahren zum Einsatz. Deshalb ist es sehr empfehlenswert, dass Sie die Datenübertragung durch SSL sichern.

6.3.1 Davfs2 (Kommandozeile)

Nutzen Sie folgenden Befehl:

```
sudo mount.davfs -o userid=<LINUX-USER> http://<Servername>:<Port>/ix-  
<Archivname>/ixwebdav/mnt/<Mount-Point>
```

Ersetzen Sie die Platzhalter.

6.3.2 Gnome/Nautilus

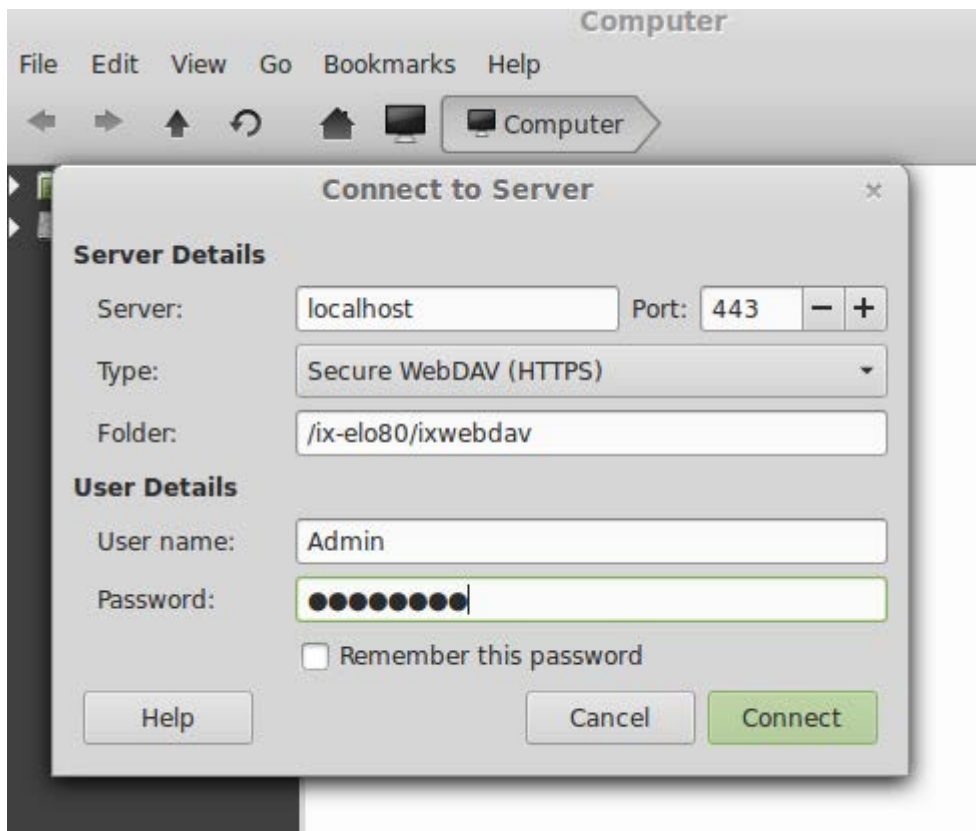


Abb. 15: Einrichtung über Nautilus

In Nautilus klicken Sie auf *Orte > Mit Server verbinden* oder auf im Menü *File* auf *Mit dem Server verbinden*.

6.3.3 KDE/Dolphin

Öffnen Sie die Adresse zum ELO WebDAV mit Dolphin oder Konqueror.

6.4 WebDAV unter MAC OS X verwenden

Unter MAC OS X wird das Authentifizierungsverfahren NTLM nicht verwendet. Stattdessen kommt das Basic-Authentication-Verfahren zum Einsatz. Deshalb ist es sehr empfehlenswert, dass Sie die Datenübertragung durch SSL sichern.

6.4.1 Kommandozeile

Nutzen Sie folgenden Befehl:

```
mount -t webdav http://<Servername>:<Port>/ix-<Archivname>/ixwebdav/<Mount-Point >
```

6.4.2 Finder

Klicken Sie auf im Finder auf *Gehe zu > Mit Server verbinden...*

Geben Sie bei *Serveradresse* die Internetadresse zum ELO WebDAV ein.

Klicken Sie auf *Verbinden*.



Hinweis: Während der Bearbeitung unter Finder werden viele temporäre Dateien erzeugt, im Archiv hochgeladen und danach wieder gelöscht. Außerdem werden die Dateien auch im Root-Ordner von ELO erzeugt. Deshalb muss der angemeldete Benutzer Schreibrechte für den Root-Ordner besitzen.

6.4.3 Cyberduck

Der Gratis Tool Cyberduck erzeugt, anders als der Finder, keine temporäre Dateien.

Nach der Installation von Cyberduck, starten Sie Cyberduck von Ihrem Dock oder vom Programmordner. Klicken Sie auf *Neue Verbindung*.

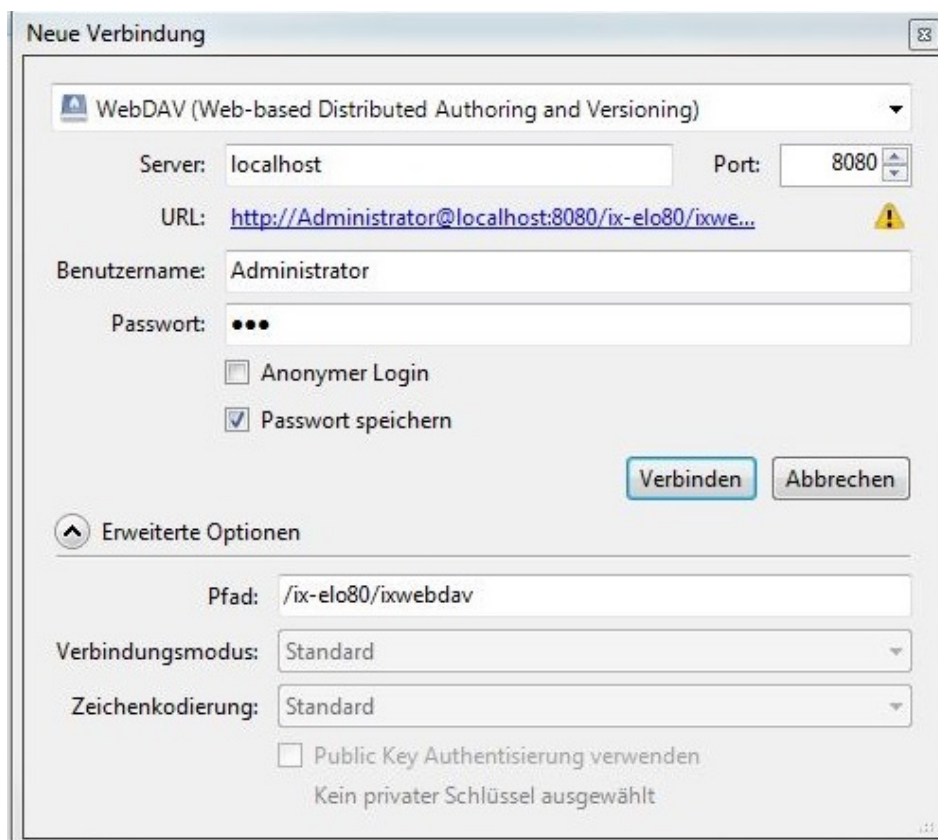


Abb. 16: Einrichtung in Cyperduck

6.5 WebDAV unter Android verwenden

6.5.1 WebDAV File Manager

Für die Verbindung mit ELO WebDAV muss zuerst ein neuer Server konfiguriert werden (Menüeintrag *Server Add*).

- Name: Name der Verbindung (z.B.: „ELO“)
- URL: Internetadresse zum ELO WebDAV.
-



Beachten Sie: Hier darf keine Port-Nummer eingegeben werden.

- User Name: ELO-Benutzername
- Password: Passwort für das ELO Konto.

Wählen Sie die Einstellung *Optionally specify a port number* und geben Sie den Port ein.

Klicken Sie *Registration*.

6.5.2 WebDAV Nav Lite

Klicken Sie auf das Plus-Zeichen zum Einfügen eines neuen WebDAV-Servers.

- Name: Name der Verbindung (z.B.: „ELO“)
- Server URL: Internetadresse zum ELO WebDAV.
- Benutzername: ELO-Benutzername
- Passwort: Passwort für das ELO Konto.

6.6 Webdav unter iOS verwenden

6.6.1 WebDAV Navigator

Klicken Sie auf das Plus-Zeichen zum Einfügen eines neuen WebDAV-Servers.

- Name: Name der Verbindung (z.B.: „ELO“)
- Server-Adresse: Internetadresse zum ELO WebDAV.
- Benutzername: ELO-Benutzername
- Passwort: Passwort für das ELO Konto.

6.6.2 Over The Air

Klicken Sie auf *Add Server*.

- Display Name: Name der Verbindung (z.B.: „ELO“)
- URL: Internetadresse zum ELO WebDAV.

- User ID: ELO-Benutzername
- Password: Passwort für das ELO Konto.

6.7 Übersicht: WebDAV-Clients

Mit folgenden WebDAV-Clients wurde ELO WebDAV getestet

Client	Betriebs- systemen	Als Laufwerk verbinden	LOCK	PROPPATCH ¹	MOVE / COPY ²	Kostenlos	Bemerkungen
Windows Explorer	Windows	Ja	Nein	Nein		N/A	
BitKinx	Windows	Nein	Ja	Nein	Ja	Ja	
WebDrive	Windows, Mac	Ja	Ja	Nein	Nein	Nein	Erzeugt Temp-Dateien
NetDrive	Windows	Ja	Nein	Nein	Nein	Ja, für nicht kommerzielle Nutzung	
Cyberduck	Windows, Mac	Nein	Nein	Ja	Nein	Ja	
Cross FTP Pro	Windows, Mac, Linux	Nein	Ja	Nein	Ja	Nein	
CarotDAV	Windows	Nein		Ja	Ja	Ja	
WebDav- plugin von TotalCom mander	Windows, Android	Nein	Nein	Nein		Kostenlos für Android	
Transmit	Mac	Laufwerk und eigene UI	Nein	Nein	Ja	Nein	Erzeugt Temp-Dateien
Finder	Mac	Ja	Nein	Nein	Nein	N/A	Temp-Dateien im Root-Ordner und im aktuellen Order speichern
Cadaver	Linux	Nein	Ja	Ja	Ja	Ja	Kommandozeile
Dolphin	Linux	Nein	Nein	Nein	Ja	Ja	
Nautilus	Linux	Nein	Nein	Nein	Nein	Ja	
davfs2	Linux	Ja	Nein	Nein	Nein	Ja	Erzeugt Temp-Dateien
Microsoft Office	Windows, Mac	Nein	Nein	Nein	Nein	Nein	

¹ Änderung der Metadaten wie Verschlagwortung

² MOVE/COPY im Server ohne Herunterladen von Dateien

WebDAV File Manager	Android	Nein	Nein	Nein	Ja	Ja	
WebDAV Nav Lite	Android	Nein	Nein	Nein	Ja	Ja	
FolderSync Lite	Android	Nein	Nein	Nein	Ja	Ja	
WebDAV Navigator	iOS	Nein	Nein	Nein	Ja	Ja	
OverTheAir	iOS	Nein	Nein	Nein		Ja	Nicht zum Bearbeiten. Nur zum Lesen und Hochladen der Dateien.

7 Dynamische Stichwortlisten

Bei dynamischen Stichwortlisten handelt es sich um Stichwortlisten, deren Inhalt zur Laufzeit erzeugt wird. Der Inhalt von dynamischen Stichwortlisten kann auch auf bereits eingegeben Daten im Verschlagwortungsdialog reagieren und die Stichworte kontextabhängig filtern.

7.1 Beispiel

Man hat in der Verschlagwortungsmaske die Felder *Abteilung* und *Mitarbeiter*. Die Definition der Abteilungen und ihrer Mitarbeiter wird in einer Datenbank außerhalb von ELO gepflegt. Um die aktuellen Abteilungen abzufragen, wird ein Skript ausgeführt, welches sich mit der externen Datenbank verbindet, vorhandene Abteilungen abfragt und als Stichwortliste zurückliefert. Aus dieser Liste kann dann die gewünschte Abteilung ausgewählt werden.

Das Feld für den Namen des Mitarbeiters sollte nun mit einem Mitarbeiter aus der zuvor ausgewählten Abteilung gefüllt werden. Hierzu sendet die Anwendung die ausgewählte Abteilung an das Skript mit der Information, jetzt die Mitarbeiter abzufragen. Der weitere Verlauf ist wie gehabt.

7.2 Administration

Ein Indexfeld erhält die neue Option *ServerScript*. Hier ist der Name des Skriptes einzutragen, welches zum Laden der Daten für die dynamische Stichwortliste ausgeführt werden soll.

Das Skript müssen Sie im Ordner *Administration* unter *IndexServer Scripting Base* hinterlegen.

7.3 Client-Anwendung

Es wird ein Objekt verschlagwortet. Im Dialog gibt der Benutzer Daten in ein Indexfeld mit hinterlegter dynamischer Stichwortliste ein. Die Client-Anwendung erkennt ein solches Indexfeld daran, dass das Feld *DocMaskLine.serverScriptName* gesetzt ist.

Sobald die Client-Anwendung eine Eingabeverzögerung erkennt, ruft diese die IX-API-Funktion

```
checkoutKeywordsDynamic(ClientInfo,KeywordsDynamicInfo)
```

auf. Im IX wird sodann das zu den gegebenen Parametern entsprechende Skript aufgerufen und das Ergebnis als

```
KeywordsDynamicResult
```

zurückgegeben. Das Ergebnis wird in der Client-Anwendung als Liste oder als Tabelle unterhalb des Indexfeldes angezeigt. Der Benutzer kann daraus die für das Objekt richtige Zeile auswählen.

Mit den Informationen aus diesem Ergebnis kann die Client-Anwendung das in Bearbeitung befindliche Indexfeld vervollständigen. Sind im Ergebnis weitere Spalten mit der Information darüber gegeben, zu welchem Schlüssel die Daten aus dieser weiteren Spalte gehören, besteht für die Client-Anwendung sogar die Möglichkeit weitere Indexfelder zu vervollständigen.

7.4 Beispiel-Skript

Im Folgenden ist ein funktionierendes Beispiel-Skript gelistet, welches zum einen die benötigten Funktionen zeigen und zum anderen als Vorlage für Skripte dienen soll. Rudimentäre Beschreibungen der Funktionen sind im Listing dokumentiert.

7.4.1 SampleScript.js

```
importPackage(Packages.de.elo.ix.jscript);

/**
 * Diese Funktion muss im Skript implementiert werden. Sie wird
 * ohne Argumente aufgerufen. Als Rückgabe wird ein neues Objekt
 * erwartet, welches die abstrakten Funktionen der Klasse
 * DynamicKeyword implementiert.
 */
function getDataIterator() {
    return new DynamicKeyword(new SimpleDatabaseQuery());
}

/**
 * Implementierung der Klasse DynamicKeyword
 * var index: Laufvariable für das ResultSet
 * var resultset: Ergebnis der DB-Abfrage
 */
function SimpleDatabaseQuery() {
    this.index = 0;
    this.resultset = undefined;
}

/**
 * Öffnet die Datenquelle. Diese Funktion wird als erstes aufgerufen.
 * @param ec: Ausführungskontext des Skripts.
 *   Enthält ec.user: UserInfo, ec.ci: ClientInfo,
 * @param sord: Das derzeit in Bearbeitung befindliche Sord. Belegte
 *   Indexfelder können als Filter für die Datenquelle dienen.
 * @param key: Schlüssel der aktuell im Client ausgewählten Zeile.
 */
SimpleDatabaseQuery.prototype.open = function(ec, sord, key) {

    // this.getObjKeyValue erleichtert den Zugriff auf Werte in Indexfeldern
    // Rückgabe ist ein Array von Strings.
    var filter = this.getObjKeyValue(sord, key);

    // SQL-Statement mit Platzhaltern. Diese werden dann von params belegt
```



```
var statement = "SELECT userid, data_1, data_2, data_3"
    + " FROM testkeywordsdynamic"
    + " WHERE userid LIKE ? AND data_1 LIKE ? ORDER BY id;";

// params[0] ersetzt das erste ? in statement, params[1] das zweite ?
var params = [];

if (ec.user.id == 0) {
    // Der Administrator soll alle Daten geliefert bekommen
    params.push("%");
} else {
    // Ein normaler User soll nur seine Daten geliefert bekommen
    params.push(ec.user.guid);
}

if (filter && filter[0]) {
    // Falls das Sord Daten für das Indexfeld 'key' bereitstellt,
    // eine Präfixsuche in der DB durchführen.
    params.push(filter[0] + "%");
} else {
    // Ansonsten nichts filtern.
    params.push("%");
}

var database = new DBConnection("jdbc/TestDataSource", "elo90");
this.resultset = database.query(statement, params);
};

/* Gibt jeweils die erste noch nicht zurückgegebene Zeile zurück */
SimpleDatabaseQuery.prototype.getNextRow = function() {
    return this.resultset[index++];
};

/* Wird aufgerufen, nachdem getNextRow() keine neuen Ergebnisse mehr
 * zurückgibt oder bereits 1000 Ergebniszeilen abgeholt wurden. */
SimpleDatabaseQuery.prototype.close = function() {
    // DB Verbindung ist bereits geschlossen, sobald database.query(...)
    // zurückkommt. Daher in diesem Fall leer.
};

/* Gibt Überschriften für Spalten als Array zurück. Die Anzahl an Spalten
 * muss mit der Anzahl an Spalten von getNextRow() übereinstimmen. Der IX
 * prüft die Anzahl nicht. */
SimpleDatabaseQuery.prototype.getHeader = function() {
    return ["Nothing", "Something", "Special", "Important"];
};

/* Gibt die Schlüssel für Spalten als Array zurück. Die Anzahl an Spalten
 * muss mit der Anzahl an Spalten von getNextRow() übereinstimmen. Der IX
 * prüft die Anzahl nicht. */
```

```
SimpleDatabaseQuery.prototype.getKeyNames = function() {
    return ["", "testKeywordScriptUser", "", ""];
};

/* Wird aufgerufen, nachdem der IX alle Zeilen abgeholt hat. Sollten aber
 * noch Zeilen übrig sein (also insgesamt >1000), gibt hasMoreRows() true
 * zurück, sonst false. */
SimpleDatabaseQuery.prototype.hasMoreRows = function() {
    return (this.index < (this.resultset.length));
};

/* Gibt eine Nachricht zurück, die der Client anzeigen soll. Können
 * Fehlermeldungen oder Informationsmeldungen sein (z.B. „Bitte zunächst
 * Feld XYZ ausfüllen“) */
SimpleDatabaseQuery.prototype.getMessage = function() {
    return "";
};

/* Gibt einen aussagekräftigen Namen für die Stichwortliste zurück
 */
SimpleDatabaseQuery.prototype.getTitle = function() {
    return "Auftragsdaten";
};
```



Beachten Sie: Die Verwendung von Variablennamen, für die im Skript eine `get...()`-Funktion existiert, kann zu einer unendlichen Rekursion führen.

Die Java-Script-Engine wandelt Zugriffe auf Properties in entsprechende Getter-Funktionen um. Dadurch entsteht im gekürzten Beispiel unten eine endlose Rekursion sobald die Funktion `getTitle()` aufgerufen wird:

```
function BadKeywordList() {
    this.title = "My Title";
}

BadKeywordList.prototype.getTitle = function() {
    // wird in this.getTitle() umgewandelt.
    return this.title;
};
```

7.5 Konfiguration in der Verschlagwortungsmaske

Das Indexserver Skript muss in der Indexzeile der Maskendefinition angegeben werden. Der Name ergibt sich über die Kurzbezeichnung des Skriptes in der *Indexserver Scripting Base*. Tragen sie in der ELO Administration Console den Skriptnamen in das Feld *Dynamische Stichwortliste* ein.

The screenshot shows the configuration interface for a dynamic keyword list. On the left, there is a sidebar with a list of fields: Abteilung, Name, Adresse, Kundennummer, Geburtsdatum, and Vorname. The main area contains configuration options for a field named 'Name'. The 'Dynamische Stichwortliste' field is highlighted with a circle and contains the text 'DynamicKeywordTest'. Other fields include 'Name', 'Gruppe' (set to 'NAME'), 'Min. Eingabelänge' (0), 'Max. Eingabelänge' (255), 'Eingabeart' (Text), 'Standardwert', 'Rechte' (Jeder), 'Externe Daten', 'Quick Info', 'Tabzuweisung' (Basis), and checkboxes for 'Neuer Tab nach diesem Indexfeld', 'Eintragungen nur mit Stichwortliste erlaubt', and 'Übersetzte Stichwortliste'. On the right, there are buttons for 'Abbrechen', 'Neues Feld', and 'Löschen'.

Abb. 17: Eintrag im Feld 'Dynamische Stichwortliste'

7.6 Dynamische Stichwortlisten im ELOWf

7.6.1 Verwendung aus Maskendefinition wie in anderen Clients

Im ELOWf können dynamische Stichwortlisten für die Verschlagwortung wie in anderen Clients verwendet werden. Dazu muss das Eingabefeld wie mit einer normalen Stichwortliste konfiguriert und der Gruppenname als Listenname verwendet werden. Voraussetzung hierfür ist, wie bei den anderen Clients, die Konfiguration des Skriptes in der Maskendefinition (siehe oben).

The screenshot shows the configuration interface for a dynamic keyword list in the ELOWf. On the left, there is a list of fields: Text1, Text2, Date1, Text2, IX_MAP_TEST1, and WF_MAP_TEST1. The main area contains configuration options for a field named 'Text'. The 'Variablenname' field is highlighted with a yellow border and contains the text 'IX_GRP_TEXT2'. Other fields include 'Stichwortliste' (Keyword), 'Gruppenname' (TEXT2), 'Autofill', 'Nur Listenwerte erlaubt', and 'Darstellung'. On the right, there is a button for 'Löschen'.

Abb. 18: Konfiguration im Formulardesigner

Wird solch ein Feld im Formular verwendet, sendet dieses wie andere Clients die aktuelle Verschlagwortung aus dem Formular als Sord an den Server und bekommt eine Tabelle mit Werten für einzelne Indexfelder.

Text1: Test2

Text2: Test2

Date1: 18.05.1977

Text2: Test2

Text1	Text2	Number1	Date1
Test2	Test2	2	23.09.1977
Test2	Test22	22	18.05.1977
Test2	Test222	222	30.09.1977
Test2	Test2222	2222	05.02.1977

Nicht weiterleiten, nur Zu...

Speichern Drucken

Abb. 19: Beispiel einer dynamischen Stichwortliste im Formular

7.6.2 Verwendung von dynamischen Stichwortlisten für Map-Felder

Im ELOwf können zusätzlich zur herkömmlichen Anwendung dynamische Stichwortlisten auch für Map-Felder definiert werden.

Variablenname: IX_MAP_KST1

Stichwortliste: Dynamic Keyword Map

Skriptname: Kostenstelle

Filter: IX_MAP_BEZ{i},IX_MAP_KTO{*},EL

☒ Autofill

☐ Nur Listenwerte erlaubt

Darstellung:

Tooltip:

Abb. 20: Einrichtung einer dynamischen Stichwortliste mit einem Map-Feld

Der Variablenname wird als `focusName` an das Indexserver Script übergeben, sobald die dynamische Stichwortliste für dieses Feld aufgerufen wird. Als Stichwortliste muss *Dynamic Keyword Map* verwendet werden. Es erscheinen die Felder *Skriptname* und *Filter*. In *Skriptname* wird der Name (Kurzbezeichnung des Indexserver Skriptes eingesetzt (bitte beachten Sie dass in diesem Fall die Methode `openMap` im Skript aufgerufen wird. Unter *Filter* können alle Felder eingetragen werden, die beim Aufrufen der dynamischen Stichwortliste in die Map übernommen und an das Script übergeben werden sollen. Auch hier gilt das Namensschema mit *IX_MAP_* und *WF_MAP_* für die Map Felder, sowie *ELO_PARAMS_* um Werte aus dem statischen *ELO_PARAMS* Objekt zu übertragen (z.B: *ELO_PARAMS_ELO_OBJID* überträgt die Objekt-Id des Eintrages). Die Map wird ohne Änderungen als Objekt `map` an das Skript übergeben.

Beim Aufrufen der dynamischen Stichwortliste wird das fokussierte Eingabefeld als `focusName` und die Map mit allen in Filter definierten Feldern inklusive dem fokussierten Feld als `map` übergeben.

7.6.2.1 Besonderheiten

Für die Funktionalität *JS_ADDLINE* können im Filterfeld Platzhalter `{i}` für den Index eingetragen werden. In diesem Fall wird beim Aufruf der dynamischen Indexzeile der Index des Fokusfeldes ausgelesen und anstelle des Platzhalters gesetzt. Wird im obigen Bild zum Beispiel für die 4. Spalte die dynamische Stichwortliste im Feld *IX_MAP_KST4* aufgerufen, wird *IX_MAP_KST4* als `focusName` an das Skript übergeben. In der Map werden die Werte für *IX_MAP_BEZ4* und *IX_MAP_KTO4* sowie der Wert des fokussierten Feldes (*IX_MAP_KST4*) übergeben.

Ebenfalls kann über den Platzhalter `{*}` alle Indexfelder an die Map übergeben werden. Als Beispiel wird im *Filter* *IX_MAP_KTO{*}* definiert und die dynamische Indexzeile von Feld *IX_MAP_KST4* aufgerufen werden für alle existierenden *JS_ADDLINE* Zeilen das Feld *IX_MAP_KTO* übertragen, also *IX_MAP_BEZ1*, *IX_MAP_BEZ2*, usw.

8 OCR über den Indexserver

Seit Version 8.00.020 kann mithilfe der Funktion „processOcr“ der „ELO OCR Service“ angesteuert werden. Voraussetzung dafür ist, dass der Service entsprechend konfiguriert ist.

8.1 Konfiguration

Das Serversetup von ELO 9 trägt soweit möglich die nötigen Konfigurationswerte ein. Allerdings richtet es nur einen OCR Worker Prozess ein, der für den ELOtr vorgesehen ist. Dies ist nicht ausreichend, wenn auch die Java-Clients die OCR über den Indexserver ausführen sollen - was automatisch dann der Fall ist, wenn auf den Client-PCs keine OCR installiert ist.

Unter dem Registrierungsschlüssel

HKEY_LOCAL_MACHINE\Software\Wow6432Node\ELO Digital\OCR\Service

kann die Anzahl der Worker im Wert „WorkerCount“ festgelegt werden. Der „ELO OCR Service“ startet aber höchstens so viele Prozesse, wie der Computer CPUs anbietet. Die Prozesse laufen standardmäßig auf niedrigster Priorität, damit sie den übrigen Betrieb möglichst wenig stören.

Die Verbindung zwischen OCR und Indexserver ist eine HTTP Verbindung. Die OCR meldet sich beim Indexserver an und wartet auf Aufgaben (Server-Push-Technik). Somit erfolgt die Festlegung, welcher „ELO OCR Service“ mit dem Indexserver arbeitet in der Konfiguration des „ELO OCR Service“ – nicht beim Indexserver.

Die Indexserver URL und das Anmeldekonto wird unter dem Schlüssel

...ELO Digital\OCR\Worker\OcrForIx\<Server-Port-Archiv>

angegeben. Wobei „<Server-Port-Archiv>“ nur als Konvention vorgeschlagen wird. Wie der Sub-Schlüssel unter „OcrForIx“ heißt, spielt technisch keine Rolle.

Unter diesem Schlüssel ist im Wert „url“ die Indexserver URL und im Wert „credentials“ das Anmeldekonto plus Semikolon plus Kennwort einzutragen.

Beispiel:

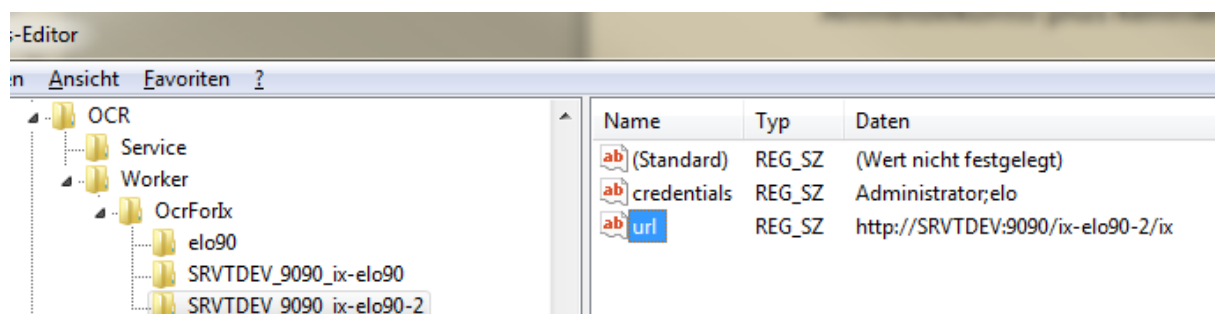


Abb. 21: Registrierungsschlüssel für OCR erstellen

Der Registry-Zweig wird vom „ELO OCR Service“ alle 10s geprüft und Änderungen werden ohne Neustart übernommen. Die Klartexteingabe des Anmeldekontos wird automatisch verschlüsselt.

Ein „ELO OCR Service“ kann mit mehreren Indexservern Verbindung aufnehmen. Andersherum kann ein Indexserver mit mehreren „ELO OCR Service“ Instanzen zusammenarbeiten. Beispielsweise könnte man für eine Anwendung, die intensiv die OCR verwendet, alle „ELO OCR Service“ Dienste, die mit dem Java-Client auf den Client-PCs installiert sind, auf den Indexserver konfigurieren.

8.2 Aufruf der processOcr Funktion

8.2.1 Feststellen, ob OCR möglich ist

Mit diesen Zeilen kann festgestellt werden, ob die OCR über den Indexserver möglich ist:

```
private bool checkOcrAvail(IXConnection conn)
{
    bool ocrAvail = true;
    try
    {
        conn.Ix.processOcr(new OcrInfo());
    }
    catch (Exception e)
    {
        if (conn.Ix.parseException(e.ToString()).exceptionType !=
            IXExceptionC.UNSUPPORTED_FUNCTION)
        {
            throw e;
        }
        ocrAvail = false;
    }
    return ocrAvail;
}
```

Wenn die OCR nicht verfügbar ist, sich also keine OCR Worker beim Indexserver angemeldet haben, wird die Exception mit dem Typ „IXExceptionC.UNSUPPORTED_FUNCTION“ geworfen.

8.3 Landessprachen ausgeben

Die von der OCR unterstützten Landessprachen können wie folgt ermittelt werden.

```
OcrInfo ocrInfo = new OcrInfo();
ocrInfo.queryLanguages = new OcrInfoQueryLanguages();

// External language names should be returned in this language
ocrInfo.messagesLanguage = OcrInfoC.MESSAGES_LANGUAGE_GERMAN;

OcrResult ocrResult = conn.Ix.processOcr(ocrInfo);
```

```
for (int i = 0; i < ocrResult.queryLanguages.externalLangs.Length; i++)
{
    Logger.instance().log("external=" +
                           ocrResult.queryLanguages.externalLangs[i]);
    Logger.instance().log("internal=" +
                           ocrResult.queryLanguages.internalLangs[i]);
}
```

Die externen Sprachnamen können in einer Benutzeroberfläche ausgegeben werden. Als Parameter für die Analyse müssen die internen Sprachnamen gesetzt werden.

8.4 OCR-Analyse synchron durchführen

Die OCR Analyse kann synchron auf die folgende Art erfolgen. Für die Erkennung eines Archivdokuments setzt man „ocrInfo.recognizeFile.objId“ auf die Objekt-ID des Dokuments. Soll eine beliebige Datei analysiert werden, muss sie zur Übergabe komplett in den Speicher gelesen und dem Element „ocrInfo.recognizeFile.imageData“ zugewiesen werden.

```
OcrInfo ocrInfo = new OcrInfo();
ocrInfo.recognizeFile = new OcrInfoRecognizeFile();
ocrInfo.recognizeFile.imageData = new FileData();
ocrInfo.recognizeFile.imageData.data = File.ReadAllBytes(fileName);
ocrInfo.recognizeFile.imageData.contentType = ".TIF";
ocrInfo.recognizeFile.recognizeLangs = new String[] { "German" };
ocrInfo.recognizeFile.timeoutSeconds = 30;
ocrInfo.recognizeFile.outputFormat = OcrInfoC.TEXT;

OcrResult ocrResult = conn.Ix.processOcr(ocrInfo);
String text = ocrResult.recognizeFile.text;
Logger.instance().log("text=" + text);
```

Solange die OCR weniger als 2min für die Analyse benötigt, ist dieser einfache Weg zu empfehlen. Dauert die Verarbeitung aber länger, dann sollte die asynchrone Ausführung gewählt werden.

8.5 OCR-Analyse asynchron ausführen

Bei der asynchronen Verarbeitung wird das OCR-Ergebnis über den Ereignisbus versendet. Dadurch hat die OCR beliebig lang Zeit, den Text zu ermitteln.

```
// Synchronisation object
private Object syncObject = ("syncObj").Clone();

// OCR Result
private OcrResult result;

private String recognizeFileAsync(IXConnection conn, String fileName,
                                   int timeoutSeconds)
{
```



```
// Open an event bus private to the session ticket and add a listener.
// OCR results are send over this bus.
EventBusApi.Bus eventBus = conn.EventBusApi.OpenEventBusChannel(
    conn.LoginResult.clientInfo.ticket);
eventBus.AddListener(EventBusC.EVENT_OCR_RESULT);

// Add a handler function for events.
conn.EventBusApi.EventBusHandler +=
    new EventBusApi.ProcessEventBusEvents(
        EventBusApi_EventBusHandler);

OcrInfo ocrInfo = new OcrInfo();
ocrInfo.recognizeFile = new OcrInfoRecognizeFile();
ocrInfo.recognizeFile.imageData = new FileData();
ocrInfo.recognizeFile.imageData.data = File.ReadAllBytes(fileName);
ocrInfo.recognizeFile.imageData.contentType = ".TIF";
ocrInfo.recognizeFile.recognizeLangs = new String[] { "English" };
ocrInfo.recognizeFile.timeoutSeconds = timeoutSeconds;
ocrInfo.recognizeFile.outputFormat = OcrInfoC.TEXT;
ocrInfo.recognizeFile.pageNo = -1; // All pages

// Set this properties for asynchronous processing:

// Event bus that receives the result
ocrInfo.busId = eventBus.Id;
// ID for the request, unique over open requests on ocrInfo.busId
ocrInfo.eventId = 123;

// Process OCR
conn.Ix.processOcr(ocrInfo);

// Wait for result
lock (syncObject)
{
    long t1 = DateTime.Now.Ticks / 10000;
    int dt = timeoutSeconds * 1000;
    while (result == null)
    {
        Monitor.Wait(syncObject, dt);
        long t2 = DateTime.Now.Ticks / 10000;
        dt -= (int)(t2 - t1);
        if (dt <= 0) throw new Exception("Timeout");
        t1 = t2;
    }
}

String text = result.recognizeFile.text;
return text;
}

// Event handler function.
```

```
private void EventBusApi_EventBusHandler(long subsId, Event[] events)
{
    foreach (Event ev in events)
    {
        if (ev.type == EventBusC.EVENT_OCR_RESULT)
        {
            lock (syncObject)
            {
                this.result = (OcrResult)AnyToObject.ToObject(ev.any);

                // Wakeup thread in recognizeFileAsync()
                Monitor.PulseAll(syncObject);
            }
        }
    }
}
```

8.6 OCR-Analyse mit Rückgabe der Zeichenpositionen

Neben der reinen Textinformation kann die OCR auch die Positionen der erkannten Zeichen zurückgeben. Hierzu setzt man als Ausgabeformat „ocrInfo.recognizeFile.outputFormat = OcrInfoC.CHAR_AND_RECT_EX“. Das Ergebnis befindet sich in „ocrResult.recognizeFile.textData.data“.

Diese Daten beginnen mit dem folgenden Vorspann:

Header:

Pos	Purpose	Value	Bytes
0	File ID	0x52434F45	4
4	Version	1	4
8	Skew Angle	Double IEEE 754	8
16	Width in Points	Integer	4
20	Height in Points	Integer	4

In den Vorspann schließen sich unmittelbar je Zeichen die folgenden Daten an:

Zeichenpositionen:

Pos	Purpose	Value	Bytes
0	UTF-16 Character	Wide Character	2
2	X-Pos. Upper Left Corner	Integer	4
6	Y-Pos. Upper Left Corner	Integer	4
10	X-Pos. Bottom Right Corner	Integer	4
14	Y-Pos. Bottom Right Corner	Integer	4
18	Confidence in Percent	Integer	4
22	Original UTF-16 Character (might be replaced due to low confidence)	Wide Character	2

Die Werte sind in der Byte-Order Little Endian ausgerichtet. Die Y-Achse zeigt in positiver Richtung nach unten. Die Koordinatenwerte sind gemessen in Bildpunkten.

Beispiel:

```
OcrInfo ocrInfo = new OcrInfo();
ocrInfo.recognizeFile = new OcrInfoRecognizeFile();
ocrInfo.recognizeFile.imageData = new FileData();
ocrInfo.recognizeFile.imageData.data = File.ReadAllBytes(fileName);
ocrInfo.recognizeFile.imageData.contentType = ".TIF";
ocrInfo.recognizeFile.recognizeLangs = new String[] { "English" };
ocrInfo.recognizeFile.timeoutSeconds = 30;
ocrInfo.recognizeFile.outputFormat = OcrInfoC.CHAR_AND_RECT_EX;
ocrInfo.recognizeFile.pageNo = 0; // First pages

// Recognize document
OcrResult ocrResult = conn.Ix.processOcr(ocrInfo);

// Put result bytes into BinaryReader
byte[] data = ocrResult.recognizeFile.textData.data;
BinaryReader rd = new BinaryReader(new MemoryStream(data));

// Read header
```

```
int magic = rd.ReadInt32();
if (magic != 0x52434F45) throw new Exception("Unrecognized format");
int version = rd.ReadInt32();
if (version != 1) throw new Exception("Unsupported version");
double skew = rd.ReadDouble();
int width = rd.ReadInt32();
int height = rd.ReadInt32();

Logger.instance().log("magic=0x" + Convert.ToString(magic, 16));
Logger.instance().log("version=" + version);
Logger.instance().log("skew=" + skew);
Logger.instance().log("width=" + width);
Logger.instance().log("height=" + height);

// Read characters
int headerSize = 4 + 4 + 8 + 4 + 4;
int charSize = 2 + 4 + 4 + 4 + 4 + 4 + 2;
int nbOfChars = (data.Length - headerSize) / charSize;
for (int i = 0; i < Math.Min(10, nbOfChars); i++)
{
    char ch = (char)rd.ReadInt16();
    int left = rd.ReadInt32();
    int top = rd.ReadInt32();
    int right = rd.ReadInt32();
    int bottom = rd.ReadInt32();
    int percent = rd.ReadInt32();
    char chOrig = (char)rd.ReadInt16();

    Logger.instance().log("char " + i + ": '" + ch + "'");
    Logger.instance().log(" rect=(" + left + "," + top + "," +
        right + "," + bottom + ")");
    Logger.instance().log(" confidence=" + percent +
        ", orig='" + chOrig + "'");
}
```

8.7 OCR-Analyse eines Rechtecks

Die OCR Analyse kann auf ein Rechteck eingeschränkt werden. Das Rechteck wird im Element „ocrInfo.recognizeFile.recognizeRects“ übergeben.

```
OcrInfo ocrInfo = new OcrInfo();
ocrInfo.recognizeFile = new OcrInfoRecognizeFile();
ocrInfo.recognizeFile.imageData = new FileData();
ocrInfo.recognizeFile.imageData.data = File.ReadAllBytes(fileName);
ocrInfo.recognizeFile.imageData.contentType = ".TIF";
ocrInfo.recognizeFile.recognizeLangs = new String[] { "English" };
ocrInfo.recognizeFile.timeoutSeconds = 30;
ocrInfo.recognizeFile.outputFormat = OcrInfoC.TEXT;
ocrInfo.recognizeFile.pageNo = 0;

ocrInfo.recognizeFile.rectUnit = OcrInfoC.UNIT_PER_THOUSAND;
```

```
int pmLeft = 50, pmTop = 100, pmRight = 200, pmBottom = 1000;
ocrInfo.recognizeFile.recognizeRects = new OcrRect[] {
    new OcrRect(pmLeft, pmTop, pmRight, pmBottom) };

OcrResult ocrResult = conn.Ix.processOcr(ocrInfo);
String text = ocrResult.recognizeFile.text;
```

Wenn die Größe des Dokuments in Bildpunkten nicht vorab bekannt ist, bietet es sich an, die Koordinaten des Rechtecks relativ zu den Dokumentabmessungen anzugeben. Dafür gibt man „ocrInfo.recognizeFile.rectUnit = OcrInfoC.UNIT_PER_THOUSAND“ an und übergibt die Koordinaten als Promille-Werte relativ zur Breite bzw. Höhe.

9 Preview-Generierung

Die Ansicht von Dokumenten in einem Web-Browser oder auf Mobilgeräten ist ohne zusätzliche Erweiterungen meist nicht möglich. Um PDF- oder TIFF-Dateien dennoch anzeigen zu können, gibt es seit Version 9.00.000 ein Modul im Indexserver, welches verschiedene Eingabeformate in Vorschaubilder konvertiert, auf eine gewünschte Größe skaliert und bei Bedarf vorhandene Annotationen zeichnet.

Die Funktionen zum Skalieren sowie Zeichnen von Annotationen können auch auf im Archiv hinterlegte Bilddokumente angewendet werden. Die unterstützten Formate sind BMP, JPEG, PNG und PSD (Adobe Photoshop).

9.1 Technischer Hintergrund

Um eine Vorschau für ein Dokument als Bilddatei zu erzeugen, sind folgende zwei Schritte notwendig:

URL-Erzeugung für die Seiten eines Dokumentes

Es wird das Dokument gelesen, die erste Seite gerendert und dem Aufrufer dessen Dimensionen, die Anzahl an Seiten des Dokumentes und für jede Seite eine generierte URL zurückgegeben.

Herunterladen der Vorschau

Sobald man die URL für eine Seite des Dokumentes erzeugt lassen hat, kann man über Sie die Vorschau herunterladen. Das verwendete Protokoll hierfür ist http.

Zum Einlesen und Rendern verschiedener Dokumenttypen werden 3rd-Party-Bibliotheken verwendet. Für PDF ist dies ICEpdf, für TIFF zum einen JAI (Java Advanced Imaging) und zum anderen TwelveMonkeys (<https://github.com/haraldk/TwelveMonkeys>), welches auch zur Verarbeitung von Photoshop-Dateien verwendet wird. Abgesehen von ICEpdf werden die Bibliotheken nicht direkt angesprochen, sondern zur Laufzeit als Plug-In für die ImageIO-API geladen.

Da es vorkommen kann, dass spezielle Dokumente in den 3rd-Party-Bibliotheken Instabilität zum Beispiel in Form von hoher Prozessor- oder Speicherlast verursachen, wurde die Konvertierung von PDF- und TIFF-Dateien in einen externen Prozess ausgelagert. Dies hat den Vorteil, den Konvertierungsprozess sicher beenden und dessen Ressourcen wieder vollständig zur Verfügung stellen zu können, ohne den Indexserver selbst zu beeinträchtigen oder neu starten zu müssen. Hierzu wird beim Start des Indexservers aus dem War-Archiv die Jar-Datei elopp.jar in ein temporäres Verzeichnis entpackt und gestartet, sobald erstmalig die Preview-Generierung eines TIFF- oder PDF-Dokumentes angefragt wird. Das Stoppen des IX beendet ebenso den externen Prozess. Zusätzlich schickt der Prozess alle 10 Sekunden ein ping-Signal an den Indexserver und beendet sich selbst, falls er nicht erreicht wird.

Auf dem Server erzeugte Bilder werden mit ihren Metadaten in einem Cache zwischengespeichert. Dieser Cache hält Objekte eine konfigurierte Zeitspanne vor und räumt selbstständig veraltete Objekte auf. Sollte die Vorschau eines Dokumentes angefragt werden, welche sich bereits im Cache befindet, so wird das eigentliche Dokument nicht mehr geladen, sondern ausschließlich die Vorschau verwendet. Eine Ausnahme bilden hier PDF-Dokumente, wenn sie in einer noch nicht zwischengespeicherten Größe angefragt werden. Hier wird dann das bereits erzeugte Bild nicht kleiner skaliert, sondern das Vorschaubild in der entsprechenden Größe neu generiert, was in einer sichtbar besseren Qualität bei geringerem Speicherverbrauch des Vorschaubildes mündet.

Die Kodierung der zu erzeugenden Bilder wählt serverseitig eine Heuristik zwischen JPEG und PNG aus. Erstere Kodierung wird bei farbigen Bildern bevorzugt, da hier die Dateigröße deutlich geringer ausfällt und letztere für Text und Schwarzweißbilder. PNGs bieten bei Text den Vorteil fehlender Kodierungs-Artefakte, bei Schwarzweißbildern unterstützt die verwendete Bibliothek pngj das Erstellen indizierter Farben, wodurch die Dateigröße stärker als bei JPEG reduziert werden kann.

Zum Skalieren von Bildern wird die Bibliothek `imgscalr` verwendet. Erzeugte Bilder werden ausschließlich kleiner skaliert.

9.2 Preview erzeugen

Für ein Dokument soll eine Vorschau erzeugt werden. Hierzu ruft die Clientanwendung die Funktion

```
checkoutPreviewImageURLs(PreviewImageInfo);
```

auf. In den Parametern werden ID des Sords oder Dokuments angegeben, sowie die gewünschte Größe der Vorschaubilder, Seitenbereich und ob Annotationen gerendert werden sollen:

```
private String getSordPreviewUrl(IXConnection conn, Sord sord) {
    PreviewImageInfo imageInfo;
    imageInfo = new PreviewImageInfo();
    imageInfo.previewSize = PreviewImageInfoC.SIZE_ORIGINAL;
    imageInfo.objectId = sord.guid;
    imageInfo.startPage = 1;
    imageInfo.endPage = 1;
    imageInfo.renderAnnotations = true;
    imageInfo.renderAnnotationsData = true;
    return conn.ix().checkoutPreviewImageURLs(imageInfo).url[0];
}
```

Als Ergebnis erhält man ein

```
PreviewImageResult
```

welches neben den URLs für die einzelnen Seiten auch Meta-Informationen über die erste angefragt Seite enthält. Hierbei ist zu beachten, dass es sich bei PDF-Dokumenten um die Abmessungen in Pixel des Dokumentes in 300dpi und bei TIFF-Dokumenten um die des Originaldokumentes handelt. Da bei TIFF-Dokumenten die horizontale und vertikale Auflösung voneinander abweichen können, werden solche im Server normiert. Um Clientseitig in der Lage zu sein, eine akkurate OCR anbieten zu können, werden daher die originalen Abmessungen zurückgegeben.

Zum Herunterladen der Vorschau kann man beliebige Software verwenden, die das http-Protokoll unterstützt. Im Client erreicht man dies zum Beispiel über:

```
IXConnection.download(String url, File file);
```

9.3 Administration

Über die Konfigurationsseite des Indexservers können für die Erzeugung von Vorschaubildern verschiedene Parameter angepasst werden. Die folgenden Abschnitte erläutern deren Funktion und Standardwerte.

9.3.1 Kontrolle des Externen Prozesses

Damit Clientanwendungen zum Beispiel im Falle Verarbeitungsfehlers nicht unbegrenzt auf eine Antwort des Servers warten, findet die Erzeugung nebenläufig statt. Der Parameter

```
imagingTimeoutProcessPageSeconds
```

gibt an, wie lange auf die Erzeugung eines Vorschaubildes gewartet werden soll. Wird die Grenze überschritten, so überträgt der Indexserver ein lokalisiertes Ausweichbild:



Dokument kann nicht verarbeitet werden.

Abb. 22: Ausweichbild

Die Verarbeitung findet allerdings weiterhin statt, sodass eine spätere Anfrage mit denselben Parametern durchaus das gewünschte Ergebnis bringen kann. Der Grund für das Überschreiten des Timeouts wird meist hohe Rechenlast auf dem Server sein. Häuft sich die Übertragung der Ausweichbilder, so sollte dieser Parameter erhöht werden.

Sollte es ein Dokument geben, welches zum Beispiel eine Endlosschleife verursachen sollte, wird zum einen dessen Verarbeitung nie beendet und zum anderen entsteht durch den Prozess eine hohe Last auf dem Server. Der Indexserver erkennt solche Situationen und sendet dann dem externen Prozess ein Signal sich selbst zu beenden. Wird dies nicht umgesetzt, wird er von außen terminiert. Die Zeitspanne, bevor der Indexserver den Prozess beenden soll wird über den Parameter

```
imagingTimeoutKillProcessSeconds
```


eingestellt. Im Standard ist sein Wert auf 60 Sekunden gesetzt. Im Gegensatz zum ersten Timeout-Parameter findet eine weitere Verarbeitung sowohl des Dokuments, welches zur Auslösung der Terminierung führt, als auch alle weiteren Dokumente, die parallel in der Warteschleife waren, zu diesem Zeitpunkt nicht mehr statt. Befinden sich im Archiv Dokumente, die eine längere Verarbeitungszeit benötigen - zum Beispiel aufgrund ihrer Größe - ist dieser Wert zu erhöhen.

Dem Verarbeitungsprozess können individuelle JVM-Parameter übergeben werden. Der Wert des Parameters

```
imagingProcessJvmArgs
```

wird ungefiltert als JVM-Parameter gesetzt. Hier kann man zum Beispiel dem Programm mehr oder weniger Arbeitsspeicher zuweisen. Ohne manuelle Anpassung wird hier bei 32-bit JVMs 1500 MB, bei 64-bit 2GB gesetzt. Bitte beachten Sie, dass dies bei 32-bit JVMs von Oracle dem Maximalwert entspricht. Der Versuch einen höheren Wert anzugeben quittiert die JVM mit sofortigem Programmabbruch.

9.3.2 Cache

Im Zwischenspeicher werden erzeugte Bilder auf dem Massenspeicher persistiert. Soweit nicht anders spezifiziert wird das Verzeichnis „cache_previews“ im Konfigurationsordner des Indexservers verwendet. Geändert wird diese Einstellung über den Parameter

```
imagingImageCacheDir
```

Die folgenden Parameter schränken die Lebenszeit von Objekten im Cache nach verschiedenen Kriterien ein. Sobald entsprechende Werte überschritten werden, wird innerhalb des Caches durch Löschen von veralteten oder überdurchschnittlich großen Einträgen aufgeräumt, bis die Grenzen wieder eingehalten werden.

Die Parameter lauten:

```
imagingImageCacheMaxEntries  
imagingImageCacheLifetimeSeconds  
imagingImageCacheSpace
```

Der Reihenfolge nach sind die Standardwerte auf 250000 Einträge, 2419200 Sekunden (28 Tage) und 21474836470 Bytes (20 GiB) gesetzt.

10 Skriptentwicklung

Für die ELO 9 Version des Indexservers kann die Entwicklungsumgebung Eclipse genutzt werden, um Indexserver-Ereignisskripte zu entwickeln. Auch das Debugging ist in dieser Umgebung möglich, sodass zum Schreiben von Code und zur Fehlersuche nicht zwischen verschiedenen Werkzeugen gewechselt werden muss.

Dies wird durch das „ELO Scripting Plug-In for Eclipse“ möglich. Es wird im Eclipse über „Help – Install New Software...“ von der URL <http://forum.elo.com/elo-scripting-plugin-update/> installiert.

Neben Indexserver-Skripten können auch Skripte für den JavaClient entwickelt werden.

Das Plugin ersetzt den Rhino Debugger, der nicht mehr verwendet werden kann.

10.1 IXConnection Klasse in Ereignisskripten

Bis zur Version 9.00.012 steht für Ereignisskripte mit dem „ix“ Objekt nur der Zugriff auf die Schnittstelle „IXServicePortIF“ bereit, und es muss in jedem Aufruf die „ClientInfo“ übergeben werden. Seit Version 9.00.014 befinden sich implizit im Kontext der Skripte auch die IXConnection-Objekte „ixConnect“ und „ixConnectAdmin“. Sie enthalten die aus dem ELOas-Skripting und der Client-Programmierung gewohnten Eigenschaften und Funktionen. Im Rechtekontext des aktuellen Benutzers werden die Funktionen von „ixConnect“ ausgeführt. Benötigt das Skript Administratorrechte, kann stattdessen „ixConnectAdmin“ verwendet werden.

Beispiel:

```
function RF_testConnectionCallAPI() {
    log.info("RF_testConnectionCallAPI(");

    var ed = ixConnect.ix().createDoc("1", "", "", EditInfoC.mbSord);
    ed.sord.name = "sord.name";

    log.info(")RF_testConnectionCallAPI=" + ed);
    return ed;
}
```

Über die IXConnection Objekte erhält man auch Zugriff auf weitere Schnittstellen, z.B. der Dokument-Feed-Schnittstelle:

```
function RF_testConnectionFeed(ec, arg) {
    log.info("RF_testConnectionFeed(");

    var objId = arg;
    var act = ixConnect.getFeedService().createAction(
        EActionType.UserComment, objId);
    act.text = "Feedaction text";
    var actionGuid = ixConnect.getFeedService().checkinAction(act,
        ActionC.mbAll);

    log.info(")RF_testConnectionFeed=" + actionGuid);
    return actionGuid;
}
```

Daneben können, wie aus der Client-Programmierung gewohnt, Dokumente hoch- und heruntergeladen werden:

```
function RF_testConnectionUpload(ec, arg) {
    log.info("RF_testConnectionUpload(");

    var tempFile = File.createTempFile("ixconn", ".txt");
    log.info("create tempFile=" + tempFile);
    var fos = new FileOutputStream(tempFile);
    fos.write(tempFile.getAbsolutePath().getBytes("UTF-8"));
    fos.close();

    var guid = arg;
    log.info("createDoc, guid=" + guid);
    var ed = ixConnect.ix().createDoc("1", "", "", EditInfoC.mbAll);
    ed.sord.guid = guid;
    ed.sord.name = "RF_testConnectionUpload-doc";

    ed.document = new Document();
    ed.document.docs = [new DocVersion()];
    ed.document.docs[0].ext = ixConnect.getFileExt(tempFile);
    log.info("checkinDocBegin, ext=" + ed.document.docs[0].ext);
    ed.document = ixConnect.ix().checkinDocBegin(ed.document);

    log.info("upload, url=" + ed.document.docs[0].url);
    ed.document.docs[0].uploadResult = ixConnect.upload(
        ed.document.docs[0].url, tempFile);

    log.info("checkinDocEnd");
    ed.document = ixConnect.ix().checkinDocEnd(ed.sord, SordC.mbAll,
        ed.document, LockC.NO);

    var objId = ed.document.objId;
    log.info("objId=" + objId);
}
```

```
tempFile["delete"]();

log.info(")RF_testConnectionUpload=" + objId);
return objId;
}

function RF_testConnectionDownload(ec, arg) {
    log.info("RF_testConnectionDownload(");

    var objId = arg;
    var sord = ixConnect.ix().checkoutDoc(objId, "", EditInfoC.mbAll,
                                          LockC.NO).getSord();

    log.info("sord=" + sord);

    var tempFile = File.createTempFile("ixconn", "." +
                                       sord.docVersion.ext);
    ixConnect.download(sord.docVersion.url, tempFile);

    var textFile = new Packages.de.elo.ix.jscript.DiscFile(
        tempFile.getAbsolutePath());
    var content = textFile.readAllText();

    tempFile["delete"]();

    log.info(")RF_testConnectionDownload=" + content);
    return content;
}
```

11 Feed-Einträge erstellen

In den Dokument-Feed können Einträge programmatisch eingefügt werden, um beispielsweise einen wichtigen Workflow-Status im Feed festzuhalten.

Automatisch generierte Feed-Beiträge sollten immer mit dem Typ „EActionType.AutoComment“ geschrieben werden.

Das folgende Beispiel zeigt das Erstellen eines Feed-Beitrags:

```
private void createFeedAction(IXConnection conn,
    Sord sord, String text) throws RemoteException {
    Action action = conn.getFeedService().createAction(
        EActionType.AutoComment, sord.guid);
    action.text = text;
    action.guid = conn.getFeedService().checkinAction(
        action, ActionC.mbAll);
}
```