ELO Indexserver Tutorial

[Stand: 28.10.2010 | Programmversion: 8.00.000]

Dieses Tutorial dient der Einführung in die Arbeit mit dem ELO Indexserver.

Inhalt

1 Tutorial

1.1 Überblick

Der IndexServer stellt Funktionen des ELO-Systems als Web-Service bereit. Die meisten ELOprofessional- und ELOenterprise-Anwendungen basieren auf seiner API: z. B. Java-Client, ELO File System, Email-Archivierung ELOXC, Web-Content-Management ELO WCM, XML-Importer, Business Logic Provider. Auch den Kunden und Business Partnern steht diese API in vollem Umfang zur Verfügung.

Die Abbildung 1 stellt grafisch dar, welche Stellung der IndexServer in der ELOprofessional- und ELOenterprise-Architektur einnimmt. Er enthält einen großen Teil der ELO-Logik, die ursprünglich nur vom ELO-Windows-Client ausgeführt wurde: Berechtigungsprüfung, Workflow-Abarbeitung, Schreiben von Replikationsinformation usw. Dazu benötigt er einen direkten Zugriff auf die Struktur- und Indexinformationen des Archivs in der SQL-Datenbank. Wie der Windows-Client schreibt und liest der IndexServer die Dokumente über den Document Manager. Logins und anwenderbezogene Funktionsaufrufe werden an den Access Manager weitergeleitet.

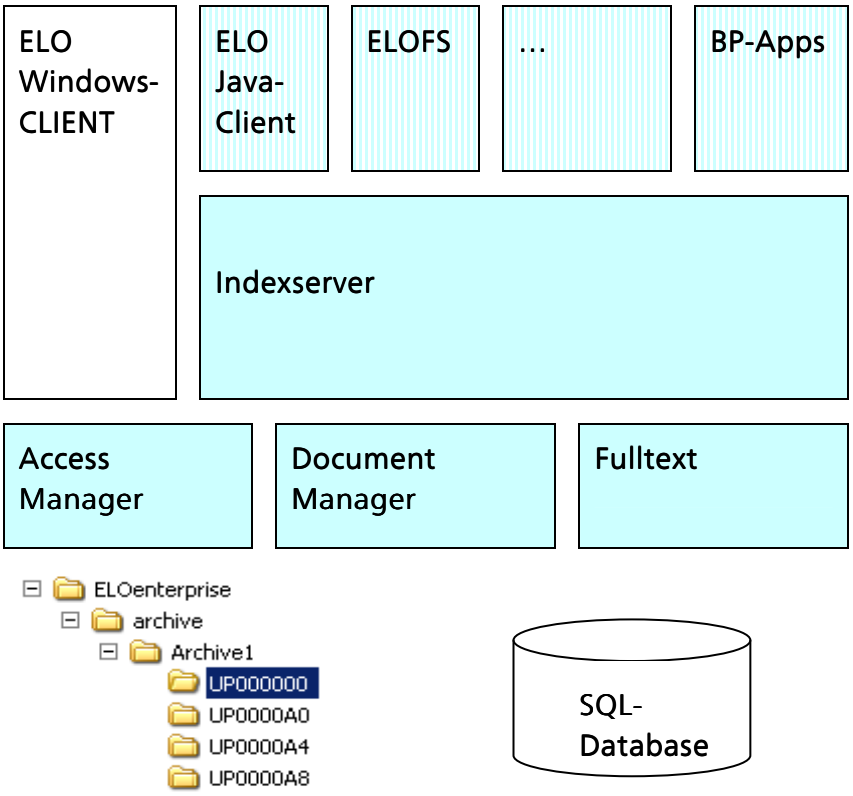


Abbildung 1: Architektur ELOprofessional/ELOenterprise

1.2 Kommunikationsprotokoll

Die IndexServer API steht sowohl für das SOAP- als auch für ein proprietäres, binäres Protokoll zur Verfügung. Das SOAP-Protokoll hat den Vorteil, dass es theoretisch von allen Betriebssystemen und Programmierumgebungen verwendet werden kann. In der Praxis sind jedoch die SOAP-Kommunikationsschichten verschiedener Programmierplattformen oft inkompatibel.

Ein weiterer Nachteil der SOAP-Kommunikation ist die starke Versionsabhängigkeit zwischen Client-Programm und IndexServer. Wenn die Indexserver-API erweitert wird, muss das Client-Programm neu übersetzt werden. Nicht zuletzt ist SOAP ein ressourcenfressendes Protokoll. Weil SOAP auf XML basiert, sind die Kommunikationsdaten recht umfangreich und müssen beim Lesen aufwendig zerlegt werden.

Um die Weiterentwicklung der Indexserver-Schnittstelle nicht wegen der Nachteile des SOAP-Protokolls einschränken zu müssen, wurde ein spezielles binäres Protokoll entwickelt. Es stellt durch eine Versionsabfrage sicher, dass ältere Client-Programme auch mit neueren Indexserver-Versionen – und umgekehrt - zusammenarbeiten können. Zudem wird es bis zu 10mal schneller verarbeitet. Auch dieses Protokoll funktioniert zwischen verschiedenen Betriebssystemen. Es ist allerdings nur für .NET und Java offiziell verfügbar, eine C++ Bibliothek kann auf Anfrage bereitgestellt werden.

Das SOAP-Protokoll ist weiterhin verfügbar, wird jedoch nur noch bei einem Hauptrelease-Wechsel an die aktuelle Schnittstelle angepasst. Wann immer möglich sollte das binäre Protokoll verwendet werden.

1.3 Verbindungsaufbau, IXConnFactory

Die Verbindung zum IndexServer wird durch eine Instanz der Klasse **`IXConnFactory`** dargestellt. Im Konstruktor ist die URL zum IndexServer zu übergeben. In einer Anwendung sollte es je angesprochenem IndexServer nur ein **`IXConnFactory`** Objekt geben.

Zur Anmeldung wird eine der **`Create`**-Funktionen aufgerufen, wie in den folgenden Beispielen gezeigt. Als Rückgabe erhält man ein **`IXConnection`**-Objekt, das die Verbindung eines Anwenders zum IndexServer darstellt.

Beispiel 1: Verbindungsaufbau, IXConnFactory

```
// Java
…
import de.elo.ix.client.*;
…
IXConnFactory connFact = new IXConnFactory(
           "http://server:8080/ix-Archive1/ix",
           "IX-Tutorial", "1.0");

IXConnection conn = connFact.create("fritz", "geheim",
           computerName, null);

System.out.println("ticket=" + conn.getLogin().ci().getTicket());

conn.logout();



// C#
…
using EloixClient.IndexServer;
…

IXConnFactory connFact = new IXConnFactory(
           "http://server:8080/ix-Archive1/ix",
           "IX-Tutorial", "1.0");

IXConnection conn = connFact.Create("fritz", "geheim",
           computerName, null);

Console.WriteLine("ticket=" + conn.Login.ci.ticket);

conn.Logout();



‘ VB.NET
…
Imports EloixClient.IndexServer
…

Dim connFact As IXConnFactory = new IXConnFactory( _
           "http://server:8080/ix-Archive1/ix", _
           "IX-Tutorial", "1.0")

Dim conn As IXConnection = connFact.Create("fritz", "geheim", _
           computerName, null)

Console.WriteLine("ticket=" + conn.Login.ci.ticket)

conn.Logout()
```

1.4 Dokument lesen

Um eine Dokumentdatei aus dem Archiv zu lesen, benötigt man die zugeordnete Identifikationskennung. Dies kann die beim Einfügen des Dokuments automatisch generierte, numerische Objekt-ID sein. Sie referenziert eindeutig ein Ordnerelement oder ein Dokument innerhalb eines Archivs.

Im Beispiel unten wird die Funktion **`checkoutDoc`** aufgerufen, um eine URL zum Dokument mit der Objekt-ID 12345 zu erhalten. Von dieser URL kann das Dokument herunter geladen werden.

Beispiel 2: Dokument lesen

```
// Java

IXConnection conn = …
String objId = "12345";

EditInfo ed = cnn.ix().checkoutDoc(objId, null,
                                   EditInfoC.mbDocument, LockC.NO);

DocVersion dv = ed.getDocument().getDocs()[0];

File outFile = File.createTempFile("ixtuto", "." + dv.getExt());

ix.download(dv.getUrl(), outFile);



// C#

IXConnection conn = …
String objId = "12345";

EditInfo ed = cnn.Ix.checkoutDoc(objId, null,
                                 EditInfoC.mbDocument, LockC.NO);

DocVersion dv = ed.document.docs[0];

String outFile = System.IO.Path.GetTempFileName() + "." + dv.ext;

ix.Download(dv.url, outFile);
```

```
 ‘ VB.NET

Dim conn As IXConnection = …
Dim objId As String = "12345"

Dim ed As EditInfo = cnn.Ix.checkoutDoc(objId, Nothing, _
                                EditInfoC.mbDocument, LockC.NO)

Dim dv As DocVersion = ed.document.docs(0)

Dim outFile As String = System.IO.Path.GetTempFileName() + "." + ext

ix.Download(url, outFile)
```

Als Zugriffskennung kann statt der Objekt-ID auch der Archivpfad oder ein Indexwert verwendet werden. Ferner ist als Zugriffskennung auch die Objekt-GUID möglich. Sie wird beim Einfügen eines Objekts erzeugt und ist mit hoher Wahrscheinlichkeit über alle Archive hinweg („weltweit“) eindeutig.

Beispiel 3: Archivpfad, Indexwert und GUID als Zugriffskennung

```
objId = "ARCPATH:/Buchhaltung/Rechnungen/2010/Hotelbuchung-11/12"
objId = "OKEY:INVOICE=4711"
objId = "(81737EC2-A845-ED5A-3EC9-AC3745A9A447)";
```

ELO erzwingt nicht, dass Archivpfade oder Indexwerte eindeutig sind. Welches Objekt bei Mehrdeutigkeit zurückgegeben wird, entscheidet der Zufall.

Neben **`checkoutDoc`** liefern die Funktionen **`checkoutSord`**, **`findFirstSords`** und **`findNextSords`** URLs für Dokumente zurück. Jedoch können sie nur die aktuelle Arbeitsversion des Dokuments bereitstellen. Die Funktion **`checkoutDoc`** hingegen kann alle Dokumentversionen und alle Attachmentversionen liefern.

1.5 Verschlagwortung lesen

Neben den Dokumentdateien gehören die Verschlagwortungsinformationen zu den wichtigsten Daten, die im ELO Archiv gespeichert sind. In der IndexServer Schnittstelle werden sie in der Klasse **`Sord`** zusammengefasst. Der Begriff kommt von Schrank-Ordner-Register-Dokument.

Das folgende Beispiel gibt einige Verschlagwortungsinformationen auf der Konsole aus. Häufig sind die Indexwerte von besonderem Interesse. Sie sind in den **`ObjKey`**-Objekten enthalten.

Beispiel 4: Verschlagwortung lesen

```
// Java
IXConnection conn = …
String objId = "12345";

EditInfo ed = conn.ix().checkoutSord(objId, EditInfoC.mbSord,
                                     LockC.NO);
Sord sord = ed.getSord();

System.out.println("id=" + sord.getId());
System.out.println("name=" + sord.getName());
System.out.println("memo=" + sord.getDesc());
System.out.println("IDate=" + sord.getIDateIso());
System.out.println("XDate=" + sord.getXDateIso());

ObjKey[] objKeys = sord.getObjKeys();
for (int i = 0; i < objKeys.length; i++) {
  ObjKey okey = objKeys[i];
  System.out.println("okey[" + i + "]:");
  System.out.println("  id=" + okey.getId());
  System.out.println("  name=" + okey.getName());
  String[] data = okey.getData();
  for (int di = 0; di < data.length; di++) {
    System.out.println("  data[" + di + "]=" + data[di]);
  }
}


// C#
IXConnection conn  = ...
String objId = "12345";

EditInfo ed = conn.Ix.checkoutSord(objId, EditInfoC.mbSord,
                                   LockC.NO);

Sord sord = ed.sord;

Console.WriteLine("id=" + sord.id);
Console.WriteLine("guid=" + sord.guid);
Console.WriteLine("name=" + sord.name);
Console.WriteLine("memo=" + sord.desc);
Console.WriteLine("IDate=" + sord.IDateIso);
Console.WriteLine("XDate=" + sord.XDateIso);

ObjKey[] objKeys = sord.objKeys;
for (int i = 0; i < objKeys.Length; i++)
{
  ObjKey okey = objKeys[i];
  Console.WriteLine("okey[" + i + "]:");
  Console.WriteLine("  id=" + okey.id);
  Console.WriteLine("  name=" + okey.name);
  String[] data = okey.data;
  for (int di = 0; di < data.Length; di++)
  {
    Console.WriteLine("  data[" + di + "]=" + data[di]);
  }
}
// VB.NET
```

```
Dim conn As IXConnection = ...
Dim objId As String = "12345"

Dim ed As EditInfo = ix.Ix.checkoutSord(objId, EditInfoC.mbSord, _
                                        LockC.NO)

Dim sord As Sord = ed.sord

Console.WriteLine("id=" + sord.id)
Console.WriteLine("guid=" + sord.guid)
Console.WriteLine("name=" + sord.name)
Console.WriteLine("memo=" + sord.desc)
Console.WriteLine("IDate=" + sord.IDateIso)
Console.WriteLine("XDate=" + sord.XDateIso)

Dim objKeys() As ObjKey = sord.objKeys
for i As Integer = 0 To objKeys.Length - 1
  Dim okey As ObjKey = objKeys(i)
  Console.WriteLine("okey[" + i + "]:")
  Console.WriteLine("  id=" + okey.id)
  Console.WriteLine("  name=" + okey.name)
  Dim data() As String = okey.data
  for di As Integer = 0 To data.Length - 1
    Console.WriteLine("  data[" + di + "]=" + data(di))
  Next di
Next i
```

Die Beschaffung aller **`Sord`**-Informationen erfordert mehrere SELECT-Statements auf der Datenbank. Um die Datenbank zu entlasten sollten nur die notwendigen Daten angefordert werden. Der Parameter **`editInfoZ`** in **`checkoutSord`** schafft die Möglichkeit, genau anzugeben, welche Daten zurückgegeben werden sollen. Er wird im folgenden Elementselektor genannt.

Im obigen Beispiel ist der Elementselektor mit **`EditInfoC.mbSord`** belegt. Damit werden alle Elemente von **`EditInfo.sord`** belegt aber keine weiteren aus **`EditInfo`**.

Das Beispiel 2 verwendet den Elementselektor **`EditInfoC.mbDocument`**. Er sorgt dafür, dass nur die Informationen zur Dokumentversion ermittelt werden.

1.6 Dokument einfügen

Das Beispiel 5 zeigt, wie ein neues Dokument ins Archiv eingefügt wird. Es geschieht in vier Schritten:

- Step 1: **`Sord`**-Objekt vorbelegen. Mit **`createDoc`** wird ein **`Sord`**-Objekt initialisiert aber noch nicht in die Archivdatenbank eingefügt. Es erbt Standardwerte vom übergeordneten Eintrag und von der Ablagemaske.

- Step 2: Dokumentversionsinformationen bereitstellen. In einem **`DocVersion`** Objekt werden die Dateiendung, der Dokumentenpfad und der Verschlüsselungskreis eingetragen. Auf der Grundlage dieser Daten erstellt **`checkinDocBegin`** eine URL zu der die Dokumentdatei hochgeladen wird.

- Step 3: Datei hochladen. Die Hilfsfunktion upload der **`IXConnection`**-Klasse übernimmt das Hochladen der Dokumentendatei. Der Adressat der URL ist standardmäßig der Document Manager. Das erspart ein erneutes Senden der Datei vom IndexServer zum Document Manger. Auf die POST Anfrage antwortet der Document Manager mit einer XML-Struktur, in der u. a. die Dokument-ID enthalten ist. Im Unterschied zur Objekt-ID, die das Dokument mit Verschlagwortungsinformationen und allen seinen Versionen kennzeichnet, referenziert die Dokument-ID eine Dateiversion.

- Step 4: **`Sord`**-Objekt einchecken. Mit dem Aufruf **`checkinDocEnd`** wird ein neues Objekt in die Datenbank eingefügt, die übergebenen Verschlagwortungsinformationen werden gespeichert und die Dokument-ID aus Step 3 mit der neu erstellten Objekt-ID verknüpft.

Beispiel 5: Dokument einfügen

```
// Java
IXConnection conn = ...
String parentId = "1";
File file = new File("c:/doc1.txt");

// Step 1
EditInfo ed = conn.ix().createDoc(parentId, "", null,
                                  EditInfoC.mbSordDocAtt);
Sord sord = ed.getSord();
sord.setName("doc1");

// Step 2
Document doc = new Document();
DocVersion dv = new DocVersion();
dv.setPathId(sord.getPath());
dv.setEncryptionSet(sord.getDetails().getEncryptionSet());
dv.setExt(conn.getFileExt(file.toString()));

doc.setDocs(new DocVersion[] {dv});
doc = conn.ix().checkinDocBegin(doc);
```

```
// Step 3
dv = doc.getDocs()[0];
String url = dv.getUrl();
String uploadResult = conn.upload(url, file);
dv.setUploadResult(uploadResult);

// Step 4
doc = conn.ix().checkinDocEnd(sord, SordC.mbAll, doc, LockC.NO);

dv = doc.getDocs()[0];
System.out.println("Object-ID=" + doc.getObjId() +
                   ", Document-ID=" + dv.getId());



// C#
IXConnection conn = ...
String parentId = "1";
String file = "c:\\doc1.txt";

// Step 1
EditInfo ed = conn.Ix.createDoc(parentId, "", null,
                                EditInfoC.mbSordDocAtt);
Sord sord = ed.sord;
sord.name = "doc1";

// Step 2
Document doc = new Document();
doc.docs = new DocVersion[] { new DocVersion() };
doc.docs[0].pathId = sord.path;
doc.docs[0].encryptionSet = sord.details.encryptionSet;
doc.docs[0].ext = conn.GetFileExt(file);

doc = ix.Ix.checkinDocBegin( doc);

// Step 3
doc.docs[0].uploadResult = conn.Upload(doc.docs[0].url, file);

// Step 4
doc = conn.Ix.checkinDocEnd(sord, SordC.mbAll, doc, LockC.NO);

Console.WriteLine("Object-ID=" + doc.objId +
                  ", Document-ID=" + doc.docs[0].id);



‘ VB.NET
Dim conn As IXConnection = …
Dim parentId As String = "1"
Dim file As String = "c:\doc1.txt"

' Step 1
Dim ed As EditInfo = conn.Ix.createDoc(parentId, "", Nothing, _
                                       EditInfoC.mbSordDocAtt)
Dim sord As Sord = ed.sord
sord.name = "doc1"

' Step 2
Dim doc As New Document()
```

```
doc.docs = New DocVersion() {New DocVersion()}
doc.docs(0).pathId = sord.path
doc.docs(0).encryptionSet = sord.details.encryptionSet
doc.docs(0).ext = conn.GetFileExt(file)

doc = conn.Ix.checkinDocBegin(doc)

' Step 3
doc.docs(0).uploadResult = conn.Upload(doc.docs(0).url, file)

' Step 4
doc = conn.Ix.checkinDocEnd(sord, SordC.mbAll, doc, LockC.NO)

Console.WriteLine("Object-ID=" + Convert.ToString(doc.objId) + _
                  ", Document-ID=" + _
                  Convert.ToString(doc.docs(0).id))
```

1.7 Dokumente suchen

Der Indexserver bietet vielfältige Möglichkeiten zum Suchen von Dokumenten an: Suche über Verschlagwortung, Volltextsuche über den Dokumentinhalt, Suche über Haftnotizen u.v.m. Das folgende Beispiel behandelt die Suche über die Indexwerte der Verschlagwortung.

Die Suche und das Abholen der Ergebnisse erfolgt in einer FindFirst-/FindNext-Schleife. Durch dieses paketweise Auslesen der Treffer wird der Servercomputer entlastet. Daneben genügt es in Client-Anwendungen mit Benutzeroberfläche häufig, dem Anwender nur die ersten x Treffer zu präsentieren und weitere erst auf Wunsch darzustellen.

Im Beispiel 6 wird eine Suche nach E-Mails gezeigt. Es werden solche Emails gesucht, die im Indexwert zur Gruppe „ELOOUTL1“ (entspricht dem Feld „Von“) mit „fritz“ und im Indexwert zur Gruppe „ELOOUTL2“ (entspricht dem Feld „An“) mit „maria“ beginnen - die Groß-/Kleinschreibung ist dabei geleichgültig. Diese Suchkriterien werden in Form eines **`FindInfo`**-Objekts beschrieben. Bei einer Suche über Verschlagwortungsdaten ist das Element **`FindInfo.findByIndex`** zu füllen. Es enthält die zu suchenden Indexwerte im Array **`FindByIndex.objKeys`**.

Der Aufruf von **`findFirstSords`** startet die Suche und sammelt (hier) bis zu 1000 Ergebnisobjekte. Der Elementselektor **`SordC.mbLean`** legt fest, dass in den Ergebnisobjekten die Indexwerte aber z. B. nicht die Archivpfade enthalten sein müssen. Die Funktion **`findFirstSords`** gibt ein **`FindResult`**-Objekt zurück, dessen Array-Element **`FindResult.sords`** die gefundenen **`Sord`**-Objekte enthält. Sie werden mit ihrer Kurzbezeichnung in **`Sord.name`** und ihren Indexwerten in **`Sord.objKeys`** auf der Konsole ausgegeben.

Wenn mit dem Aufruf von **`findFirstSords`** bereits alle Emails zurückgegeben wurden, die im Archiv gespeichert sind, dann ist **`FindResult.moreResults = false`** gesetzt und die Schleife wird beendet.

Andernfalls wird **`findNextSords`** aufgerufen, um weitere Ergebnisse zu lesen. Der Funktion muss als erstem Parameter die Such-ID übergeben werden, denn es können beliebig viele Suchen gleichzeitig laufen. Sie wurde in **`findFirstSords`** generiert und in **`FindResult.searchId`** an die Client-Anwendung zurückgegeben. Zu dieser Such-ID speichert der Indexserver eine Liste von der Objekt-IDs der gefundenen Objekte. Mit **`findNextSords`** kann auf diese Liste wahlfrei zugegriffen werden.

Zum Abschlusse einer Such-Schleife sollte immer **`findClose`** aufgerufen werden, damit der Indexserver den für die Objekt-ID-Liste belegten Speicher freigeben kann. Wenn es versäumt wird, dann gibt der Indexserver die Liste nach einer vorkonfigurierten Zeit von selbst wieder frei, standardmäßig 5 Minuten.

Beispiel 6: Dokumente suchen

```
// Java
IXConnection conn = …

FindInfo fi = new FindInfo();
FindByIndex fx = new FindByIndex();
fi.setFindByIndex(fx);

ObjKey[] okeys = new ObjKey[2];
okeys[0] = new ObjKey();
okeys[0].setName("ELOOUTL1");
okeys[0].setData(new String[] {"fritz*"});
okeys[1] = new ObjKey();
okeys[1].setName("ELOOUTL2");
okeys[1].setData(new String[] {"maria*"});

fx.setObjKeys(okeys);
fx.setMaskId("Email");

FindResult fr = null;

try {
  int idx = 0;

  fr = conn.ix().findFirstSords(fi, 1000, SordC.mbLean);
  while (true) {

    for (Sord sord : fr.getSords()) {
      ObjKey[] skeys = sord.getObjKeys();
      System.out.println("name=" + sord.getName() +
                         ", from=" + skeys[0].getData()[0] +
                         ", to=" + skeys[1].getData()[0]);
    }

    if (!fr.isMoreResults()) break;

    idx += fr.getSords().length;

    fr = conn.ix().findNextSords(fr.getSearchId(), idx, 1000,
                                 SordC.mbLean);
  };
}
```

```
finally {
  if (fr != null) {
    conn.ix().findClose(fr.getSearchId());
  }
}
```

```
// C#
IXConnection conn = …

FindInfo fi = new FindInfo();
FindByIndex fx = new FindByIndex();
fi.findByIndex = fx;

ObjKey[] okeys = new ObjKey[2];
okeys[0] = new ObjKey();
okeys[0].name = "ELOOUTL1";
okeys[0].data = new String[] { "fritz*" };
okeys[1] = new ObjKey();
okeys[1].name = "ELOOUTL2";
okeys[1].data = new String[] { "maria*" };

fx.objKeys = okeys;
fx.maskId = "Email";

FindResult fr = null;

try
{
    int idx = 0;
    fr = conn.Ix.findFirstSords(fi, 1000, SordC.mbLean);
    while (true)
    {
        foreach (Sord sord in fr.sords)
        {
            Console.WriteLine("name=" + sord.name +
                ", from=" + sord.objKeys[0].data[0] +
                ", to=" + sord.objKeys[1].data[0]);
        }

        if (!fr.moreResults) break;

        idx += fr.sords.Length;
        fr = conn.Ix.findNextSords(fr.searchId, idx, 1000,
                                   SordC.mbLean);
    }
}
finally
{
    if (fr != null)
    {
        conn.Ix.findClose(fr.searchId);
    }
}
```

```
‘ VB.NET
Dim conn As IXConnection = ...

Dim fi As FindInfo = New FindInfo()
Dim fx As FindByIndex = New FindByIndex()
fi.findByIndex = fx

Dim okeys(1) As ObjKey
okeys(0) = New ObjKey()
okeys(0).name = "ELOOUTL1"
okeys(0).data = New String() {"fritz*"}
okeys(1) = New ObjKey()
okeys(1).name = "ELOOUTL2"
okeys(1).data = New String() {"maria*"}

fx.objKeys = okeys
fx.maskId = "Email"

Dim fr As FindResult = Nothing

Try
    Dim idx As Integer = 0
    fr = conn.Ix.findFirstSords(fi, 1000, SordC.mbLean)

    While True

        For Each sord As Sord In fr.sords
            Console.WriteLine("name=" + sord.name + _
                        ", from=" + sord.objKeys(0).data(0) + _
                        ", to=" + sord.objKeys(1).data(0))
        Next

        If Not fr.moreResults Then
            Exit While
        End If

        idx += fr.sords.Length
        fr = conn.Ix.findNextSords(fr.searchId, idx, 1000,
                                   SordC.mbLean)

    End While
Finally

    If Not (fr Is Nothing) Then
        conn.Ix.findClose(fr.searchId)
    End If

End Try
```

Das Auflisten von Untereinträgen eines Ordners ist eine besondere Form der Suche und erfolgt mit Hilfe der Klasse **`FindChildren`**.

2 Erstellen von Anwendungen, benötigte Bibliotheken

Für das Erstellen von Indexserver-Programmen sind die hier aufgeführten Voraussetzungen nötig.

2.1 Java-Anwendungen

- Java Entwicklungsumgebung
- IndexServer_Programming.zip aus dem Support-Archiv
- Java 1.4: alle Dateien aus dem Verzeichnis IndexServer_Programming.zip/Java-1.4/lib
- Ab Java 1.5: alle Dateien aus dem Verzeichnis IndexServer_Programming.zip/Java-1.5/lib

2.2 .NET-Anwendungen

- Visual Studio 2005 oder neuer
- IndexServer_Programming.zip/.NET/lib/EloixClientCS.DLL als Projektverweis aufnehmen

2.3 Schnittstellenklassen einbinden

Die Schnittstelle des Indexservers kennt zwei Arten von Klassen:

- **Werteklassen** – sie enthalten Datenelemente aber keine Funktionen. Beispielsweise enthält die Klasse **`Sord`** die Daten der Verschlagwortung eines Aktenstrukturelements. Sie bietet aber keine Funktionen zum Lesen oder Speichern an. Diese Funktionen findet man in der Funktionsklasse **`IXServicePortIF`**.
- **Funktionsklasse(n)** – sie enthalten Funktionen aber keine Datenelemente. Der IndexServer bietet nur eine Klasse – genauer ein Interface – dieser Art an: **`IXServicePortIF`**. Dieses Interface enthält alle Funktionsaufrufe des Indexservers und benutzt zum Datentransport die Werteklassen.

Das folgende Beispiel skizziert die Definitionen der Werteklasse **`Sord`** und der Funktionsklasse **`IXServicePortIF`**.

Beispiel 7: Codeausschnitt für die Definitionen der Werteklasse Sord und das Interface IXServicePortIF

```
public class Sord
{
  protected int idField;
  protected String nameField;
  protected String iDateIsoField;
  …
  public int id
  {
    get { return idField; }
    set { idField = value; }
  }
  …
}

public interface IXServicePortIF {
  …
  int checkinSord(ClientInfo ci, Sord sord, SordZ sordZ,
                  LockZ lockZ);
  …
}
```

Alle Klassen der Indexserver-Schnittstelle befinden sich für Java-Anwendungen im Paket **`de.elo.ix.client`**. Microsoft .NET-Programme binden die Klassen durch „using" bzw. „Import" des Pakets **`EloixClient.IndexServer`** ein.

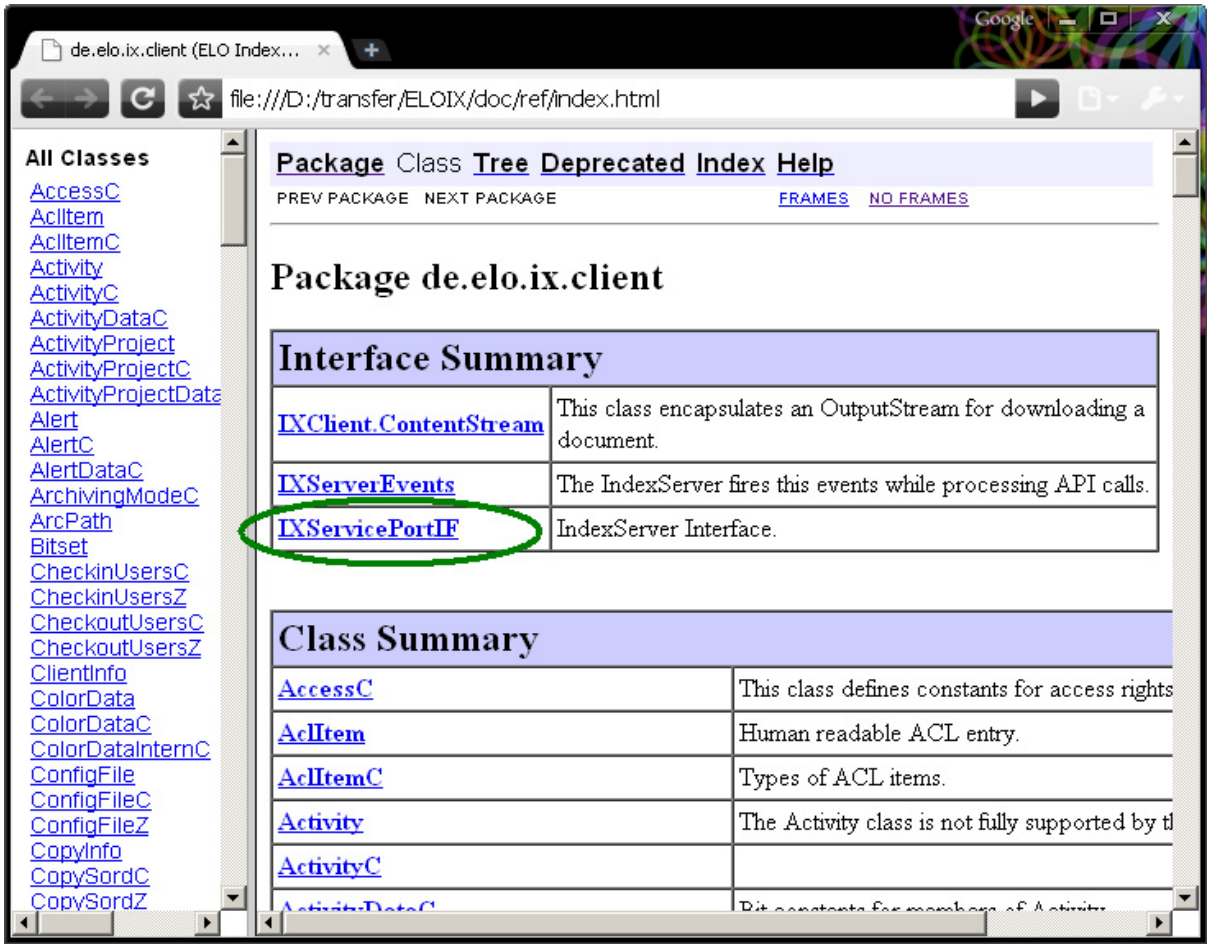
Beispiel 8: Indexserver-Symbole einbinden

```
// Java
package my.pack;
import de.elo.ix.client.*;



// C#
using System;
using System.Collections.Generic;
using System.Text;
using EloixClient.IndexServer;



‘ VB.NET
Imports EloixClient.IndexServer
```

3 Verwendung der Referenzdokumentation

Im Paket IndexServer_Programming.zip wird im Verzeichnis doc\ref\ eine mit Hilfe des Tools javadoc erstellte Referenzdokumentation der Indexserver-Schnittstelle bereitgestellt. Sie besteht aus einer Sammlung von HTML-Dateien und kann mit jedem Browser angezeigt werden.

Der Einstiegspunkt ist die Datei index.html. Zur Indexserver-Schnittstelle gelangt man über den Link „IXServicePortIF".



Wenn wie im Tutorial und in allen Beispielen dieses Dokuments die **`IXConnection`**-Klasse verwendet wird, dann entfallen bei allen Funktionen aus **`IXServicePortIF`** die **`ClientInfo`**-Parameter.