ELO Indexserver Tutorial

[Date: 10/28/2010 | Program version: 8.00.000]

This tutorial serves as the introduction for working with the ELO Indexserver.

Contents

1 Tutorial

1.1 Overview

The Indexserver provides functions of the ELO system as a web service. Most ELOprofessional and ELOenterprise applications are based on its API: such as the Java Client, ELO File System, e-mail archiving ELOXC, web content management ELO WCM, XML importer and business logic provider. This API is also available in full to the customers and business partners.

Figure 1 shows graphically which position the Indexserver has in the ELOprofessional and ELOenterprise architecture. It contains a large part of the ELO logic that originally was only executed by the ELO Windows Client: Rights check, processing workflows, writing replication information, etc. For this, the client needs direct access to the structure and index information of the archive and in the SQL database. Just as the Windows Client, the Indexserver writes and reads the documents via the Document Manager. Logins and user-related function calls are forwarded to the Access Manager.

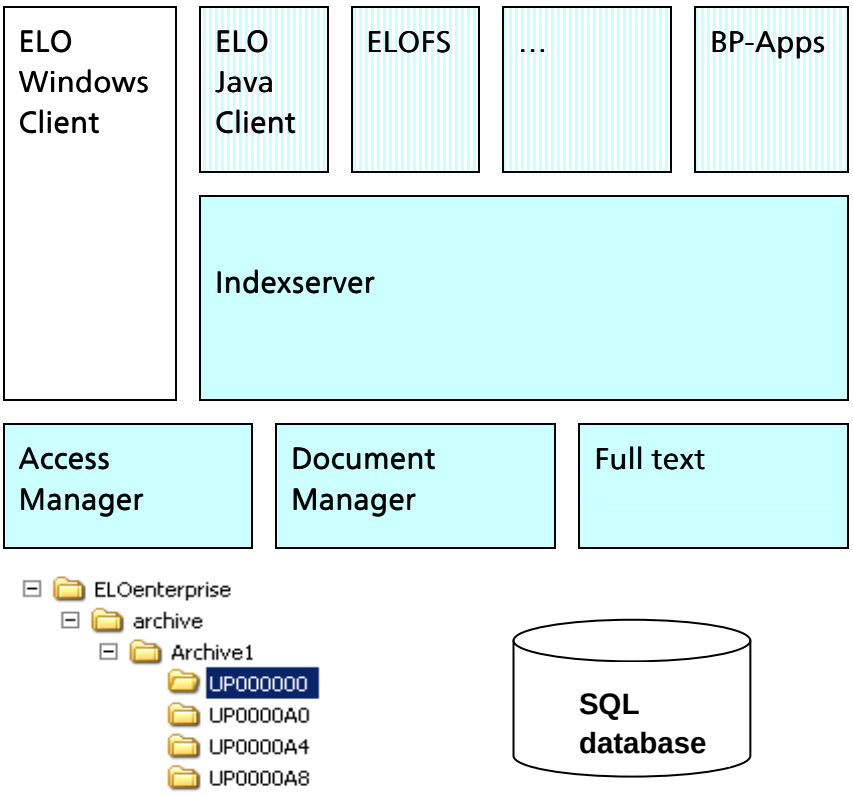


Figure 1: Architecture ELOprofessional/ELOenterprise

1.2 Communication protocol

The Indexserver API is both available for the SOAP as well as for a proprietary, binary protocol. The SOAP protocol has the advantage that it theoretically can be used by all operating systems and programming environments. In practice, however, the SOAP communication levels of different programming platforms are often incompatible.

An additional drawback of SOAP communication is the strong version-dependency between the client program and the Indexserver. When the Indexserver API is expanded, the client program must be retranslated. Finally, SOAP is a resource-hungry protocol. Since SOAP is based on XML, the communication data is quite extensive and needs to be split with a large amount of effort during reading.

To not have to limit the further development of the Indexserver interface because of the disadvantages of the SOAP protocol, a special binary protocol was developed. It guarantees through a version request that older client programs can also work together with newer Indexserver versions and vice versa. In addition, it is processed up to ten times faster. This protocol also works for different operating systems. However, it is only officially available for .NET and JAVA; a C++ library can be provided upon request.

The SOAP protocol is still available, but is only adjusted to the current interface for a main release change. You should use the binary protocol whenever possible.

1.3 Establishing the connection, IXConnFactory

The connection to the Indexserver is represented by an instance of the **`IXConnFactory`** class. The URL for the Indexserver must be transmitted in the constructor. In an application, there should only be one **`IXConnFactory`** object for each addressed Indexserver.

For the logon, a **Create** function is invoked, as shown in the following examples. As a return, you receive an **IXConnection** object that represents the connection of a user to the Indexserver.

Example 1: Establishing the connection, 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", "secret",
           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", "secret",
           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", "secret", _
           computerName, null)

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

conn.Logout()
```

1.4 Read document

To read a document file from the archive, you need the object identifier. This can be an numerical object ID that is automatically generated while adding the document. It uniquely references a folder element or a document within an archive.

In the example below, the **`checkoutDoc`** function is called to get an URL for a document with the object ID 12345. The document can be downloaded from this URL.

Example 2: Read document

```
// 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)
```

You can also use an index value instead of the object ID of the archive as the access ID. In addition, the access ID can also be the object GUID. It is generated when inserting an object and it is very likely that it is unique across all archives ("worldwide").

Example 3: Archive path, index value and GUID as access ID

```
objId = "ARCPATH:/Accounting/Invoices/2010/Hotel-reservation-11/12"
objId = "OKEY:INVOICE=4711"
objId = "(81737EC2-A845-ED5A-3EC9-AC3745A9A447)";
```

ELO does not force you to use unique archive paths or index values. Coincidence decides which object is returned in the case of ambiguity.

In addition to **`checkoutDoc`**, the functions **`checkoutSord`**, **`findFirstSord`** and **`findNextSords`** return the URLs for documents. However, you can only provide the current working version of the document. The **`checkoutDoc`** function, however, can return all document versions and all attachment versions.

1.5 Read keywording

In addition to the document files, the keywording information is among the most important data that is stored in the ELO archive. This information is summarized in the **`Sord`** class in the Indexserver interface. The term comes from Schrank-Ordner-Register-Dokument (cabinet-folder-tab-document).

The following example provides keywording information on the console. The index values often are of particular interest. They are contained in the **`ObjKey`** objects.

Example 4: Read keywording

```
// 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
```

Organizing all **Sord** information requires multiple SELECT states in the database. To put less strain on the database, only the necessary data should be requested. The parameter **editInfoZ** in **checkoutSord** provides the option of specifying which data should be returned. In the following, it will be called element selector.

In the example above, the element selector is assigned with **EditInfoC.mbSord**. With this, all elements of **EditInfo.sord** are assigned but no others from **EditInfo.**

Example 2 uses the element selector **EditinfoC.mbDocument**. It makes sure that only information on the document version is provided.

1.6 Insert document

Example 5 shows how a new document is added to the document. This is done in four steps:

- Step 1: Preassign **`Sord`** object. With **`createDoc`**, a **`Sord`** object is initiated but not yet added to the archive database. It inherits default values from the parent entry and from the keywording form.
- Step 2: Provide the document version information. The file ending, the document path and the encryption key are entered in a **`DocVersion`** object. Based on this information, **`checkinDocBegin`** creates a URL to which the document file is uploaded.
- Step 3: Upload file. The help version upload of the **`IXConnection`** class takes on the uploading of the document. The default addressee of the URL is the Document Manager. This avoids having to save the file again from the Indexserver to the Document Manager. The Document Manager answers the POST request with an XML structure in which the document ID is included. Compared to the object ID that indicates the document with keywording information and all of its version, the document ID only references one file version.
- Step 4: Check in **`Sord`** object. With the **`checkinDocEnd`** request, a new object is added to the database, the transmitted keywording information is stored and the document ID from step 3 is connected with the newly created object ID.

Example 5: Insert Document

```
// 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 Searching for documents

The Indexserver offers numerous options for searching for documents: Search via keywording, full text search via the document content, search via sticky notes, and much more. The following example is about the search via index values of the keywording.

Searching and fetching results takes place in a FindFirst/FindNext loop. Obtaining the results in packets relieves the server computer. In addition, it is often enough in applications with an interface to only display the first x results and to only display more upon request.

In example 6, a search for e-mails is displayed. E-mails that have an index value for the group "ELOOUTL1" (corresponds to the "From" field") with "tom" and start with "mary" in the index value for the group "ELOOUTL2" (corresponds to the "To" field) – capitalization is not important for this. These search criteria are described in the form of a **`FindInfo`** object. For a search via keywording information, the **`FindInfo.findByIndex`** element must be filled in. It contains the index values to be searched in the **`FindByIndex.objKeys`** array.

The **`findFirstSords`** request starts the search and gathers up to 1,000 result objects (here). The **`SordC.mbLean`** element selector defines that the event objects must include the index values but, for example, not the archive paths. The **`findFirstSords`** function returns a **`FindResult`** object whose array element **`FindResult.sords`** contains the **`Sord`** objects found. The objects are returned with their short name in **`Sord.name`** and its index values in Sord.objKeys on the console.

If already all e-mails that are stored in the archive were returned with the first **`findFirstSords`** request, then **`FindResult.moreResults`** = false is set and the loop is ended.

Otherwise, **`findNextSords`** is called up to read more results. The search ID must be given to the function as the first parameter since any number of searches can run at the same time. The ID was generated in **`findFirstSords`** and returned to the client application in **`FindResult.searchId`**. The Indexserver saves a list of the object ID of the objects found for this search ID. With **`findNextSords`**, you can access this list without having to select it.

For the end of a search loop, you should always call up **`findClose`** so that the Indexserver can release the memory used for the object ID list. If you neglect to do this, then the Indexserver releases the list after a predefined time, which by default is five minutes.

Example 6: Search documents

```
// 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
```

Listing child entries of a folder is a special form of searching and is performed with the help of the **`FindChildren`** class.

2 Creating applications, required libraries

To create Indexserver programs, the listed requirements are necessary.

2.1 Java applications

- Java development environment
- IndexServer_Programming.zip from the support archive
- Java 1.4: all files from the directory IndexServer_Programming.zip/Java-1.4/lib
- Starting with Java 1.5: all files from the directory IndexServer_Programming.zip/Java-1.5/lib

2.2 .NET applications

- Visual Studio 2005 or newer
- Add IndexServer_Programming.zip/.NET/lib/EloixClientCS.DLL as a project reference

2.3 Include interface classes

Value classes - they contain data elements but no functions.

- **Value classes** - they include data elements but no functions. For example, the **`Sord`** class contains information of the keywording of a file structuring element. However, it does not offer functions for reading or saving. These functions can be found in the **`IXServicePortIF`** function class.
- **Function class(es)** - they contain functions but no data elements The Indexserver offers only one class – or interface– of this type: **`IXServicePortIF`**. This interface includes all function calls of the Indexserver and uses the value classes for data transport.

The following example outlines the definitions of the **`Sord`** value class and the **`IXServicePortIF`** function class.

Example 7: Code excerpt for the definitions of the Sord value class and the IXServicePortIF interface

```
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);
  …
}
```

All classes of the Indexserver interface are in the Java applications in the package **`de.elo.ix.client`**. Microsoft .NET programs include the classes by "using" or "importing" the **`EloixClient.IndexServer`** packet.

Example 8: Including Indexserver icons

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

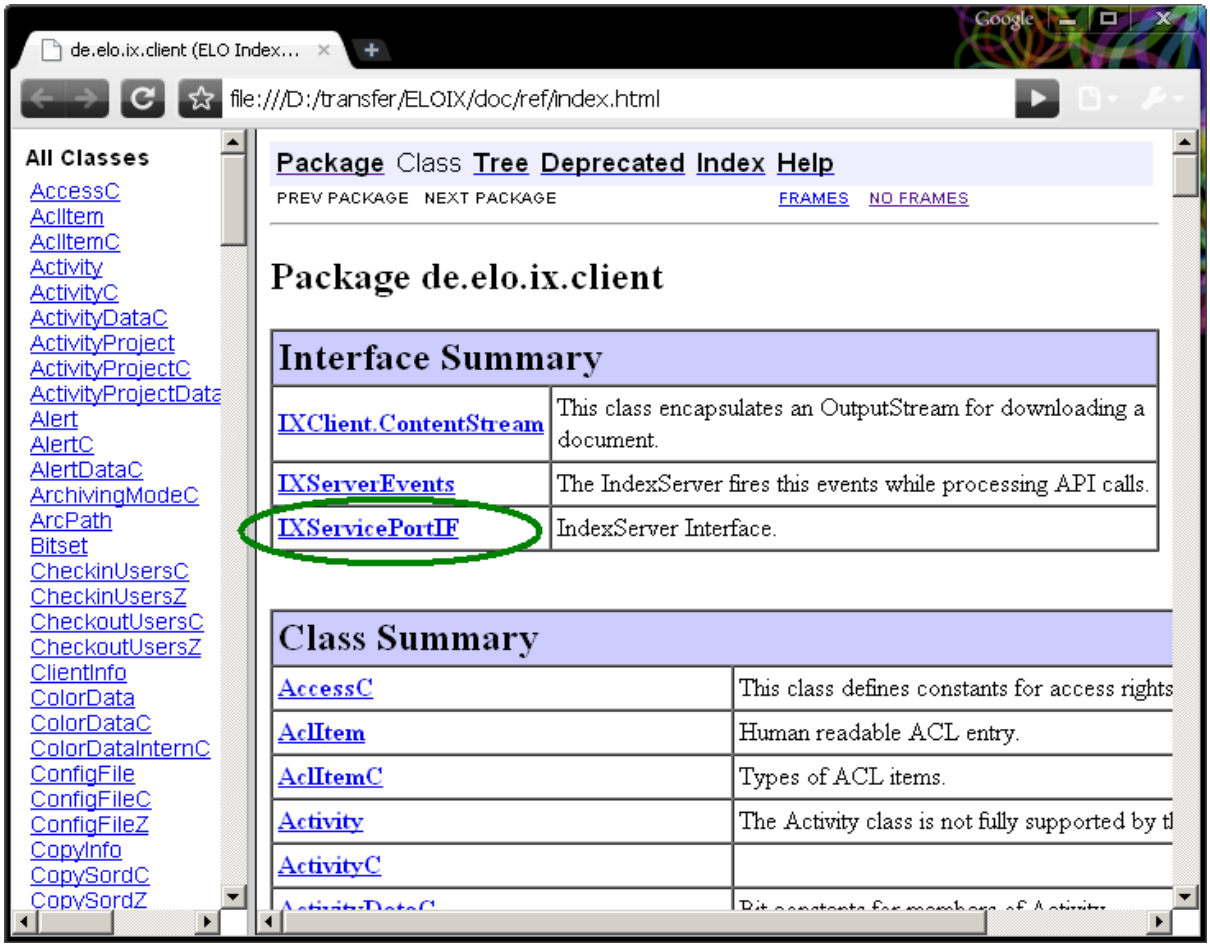
```
// C#
using System;
using System.Collections.Generic;
using System.Text;
using EloixClient.IndexServer;
```

```
‘ VB.NET
Imports EloixClient.IndexServer
```

3 Using the reference documentation

In the IndexServer_Programming.zip file, a reference document created with the javadoc tool for the Indexserver interface is provided in the doc\ref\ directory. It consists of a collection of HTML files and can be displayed with every browser.

The entry point is the file index.html. You get to the Indexserver interface via the link "IXServicePortIF".



If the **`IXConnection`** class is used as in the tutorial and all examples of this document, then the **`ClientInfo`** parameters are left out for all functions from the **`IXServicePortIF`**.