# Ereignisschnittstelle des IndexServers

Beim Einsatz des IX kann es von Nutzen sein, bei jedem Schreiben eines Dokuments server-seitig eine Funktion auszuführen, die zum Beispiel projektspezifische Plausibilitätsprüfungen durchführt oder einen Workflow startet. Solche Ereignisse können zum Einen in JavaScript-Funktionen ausgewertet werden.  Zum Anderen besteht die Möglichkeit, die Ereignisse per HTTP(S) zu einem WebService zu senden.

## 1 JavaScript Ereignisse

### 1.1 Ereignisfunktionen

Unter den JavaScript Funktionen zur Ereignisbehandlung wird zwischen synchronen und asynchronen Funktionen unterschieden. Ihnen stehen die folgenden, globalen Objekte zur Verfügung.

#### 1.1.1 Globale Objekte

| Objektname | Typ | Bedeutung |
|---|---|---|
| ix | IXServicePortIF | IndexServer Schnittstelle |
| CONST | IXServicePortC | IndexServer Konstanten-Objekt. Nur Lesezugriff erlaubt. |
| log | Log | Logger Objekt, s. Apache Commons Logging |
| globalScope | Objekt | In diesem JavaScript Objekt können beliebige Werte gespeichert werden, die jeder Skriptfunktion zur Verfügung stehen sollen. Der Inhalt geht beim Stoppen des IndexServers verloren. |
| scriptScope | Objekt | In diesem JavaScript Objekt können beliebige Werte gespeichert werden,  die nur den Funktionen innerhalb eines Skriptes sichtbar sein sollen. Der Inhalt geht beim Stoppen des IndexServers verloren. |
| threadScope | Objekt | In diesem JavaScript Objekt können beliebige Werte gespeichert werden,  die nur dem aktuellen Thread sichtbar sein sollen. Der Inhalt geht beim Stoppen des IndexServers verloren. |

Tabelle 1, Globale Objekte

#### 1.1.2 Ausführungskontext

Jeder Ereignisfunktion wird im ersten Parameter der Ausführungskontext übergeben. Er heißt per Konvention „ec“ und stellt die Informationen nach Tabelle 2, Ausführungskontext, bereit.

| Elementname | Typ | Bedeutung |
|---|---|---|
| user | UserInfo | Anwenderdaten, z.B. ec.user.name ist der Anwendername. Nur Lesezugriff erlaubt. |
| ci | ClientInfo | Sitzungsinformationen, z.B. Session-Ticket in ec.ci.ticket. Nur Lesezugriff erlaubt. |
| url | String | IndexServer URL. |

Tabelle 2, Ausführungskontext

### 1.1.3 Synchrone Ereignisfunktionen

Die synchronen Ereignisfunktionen werden im selben Thread ausgeführt, in dem die Abarbeitung der IndexServer-Schnittstellenfunktion erfolgt. Sie können zusätzliche Prüfungen durchführen, Daten ergänzen oder die Weiterverarbeitung durch Auslösen einer Exception abbrechen.

Damit die Verarbeitungsgeschwindigkeit des IndexServes nicht zu stark beeinträchtigt wird, sollten synchrone Ereignisfunktionen möglichst schnell durchlaufen werden. Außerdem sollte man sich darüber bewusst sein, dass bei Programmierfehlern alle IndexServer-basierten Anwendungen gestört werden.

Beispiele für synchrone Ereignisfunktionen sind onBeginCheckinSord und onBeginCheckinDocEnd.

Innerhalb offener Transaktionen werden synchrone Ereignisfunktionen nicht aufgerufen, damit keine Deadlocks entstehen können, wenn in den Ereignisfunktionen IndexServer-API-Funktionen aufgerufen werden. Beispielsweise wird aus diesem Grund das Ereignis onBeginCheckinSord nicht in checkinSordPath ausgelöst. Das Gleiche gilt beim Anlegen des Pfades auf Basis der Ablagemaskendefinition.

### 1.1.4 Asynchrone Ereignisfunktionen

Im Gegensatz zu synchronen Ereignisfunktionen werden asynchrone Ereignisfunktionen in einem Hintergrund-Thread verarbeitet. Sie eignen sich zur Versendung von Informationen an externe Systeme oder zum nachträglichen Ergänzen von Daten.

Asynchrone Ereignisfunktionen können die Verarbeitung der Schnittstellenfunktion im IndexServer nicht durch eine Exception stoppen. Wenn sie einen Fehler produzieren, erscheint lediglich eine Warnung in der Log-Datei.

Weil diese Ereignisse keinen direkten Einfluss auf die IndexServer Performance haben, können auch aufwendigere Operationen durchgeführt werden.

Für die Ausführung asynchroner Ereignisfunktionen steht ein Threadpool bereit, dessen Kapazität in den IndexServer Optionen eingestellt werden kann und standardmäßig maximal 5 Threads erlaubt.

Beispiele für synchrone Ereignisfunktionen sind onAfterCheckinSord und onAfterCheckinDocEnd.

### 1.1.5 Parameter der Ereignisfunktionen, gültige Datenelemente

Die in den Parametern übergebenen Objekte sind nicht vollständig gefüllt. Das bedeutet z. B., dass in einem übergebenen Sord Objekt das Element refPaths zuerst durch einen IndexServer-Aufruf nachgelesen werden muss, bevor es verwendet werden kann. Einige bestimmte Datenelemente sind dagegen immer verfügbar. Dazu gehört z.B. die Sord.id und Sord.type. Eine genaue Auflistung liefert die Referenzdokumentation.

Welche Daten im Objekt gültig sind, kann aus dem Element changedMembers ermittelt werden. Es ist ein 64bit Integer in dem jedes Bit einem Datenelement des Objekts zugeordnet ist. Wenn ein Bit gesetzt ist, dann ist das zugehörige Datenelement gültig. Die Definition der Bits z.B. für die Klasse Sord erfolgt in der Klasse SordC. Das Element Sord.name gehört zum Bit SordC.mbName, das Element Sord.objKeys gehört zum Element SordC.mbObjKeys usw.

Im JavaScript Code prüft man auf die folgende Art, ob ein Bit gesetzt ist:

```
var succ = Bitset.isTrue(sord.changedMembers, SordC.mbRefPaths);
```

Weil JavaScript keine bit-weisen Operatoren für 64bit Integer bereitstellt, wird das Hilfsobjekt Bitset hier benötigt.

## 1.2 Speicherort der Skripte

Analog zum JavaClient liegen die Skripte in einem Archivordner. Für die IndexServer Skripte ist dies der Ordner „IndexServer Scripting Base“ im Ordner „Administration“.

Darunter befindet sich ein Ordner „_ALL“, der Skripte enthält, die von jeder IndexServer-Instanz ausgeführt werden. Diese Skripte werden auch von der „ELO Archive Link“ ausgeführt.

Daneben kann es je IndexServer-Instanz weitere Ordner geben, die jeweils nur für die zugehörige Instanz gelten. Der Ordnername ist dabei gleich dem IndexServer- Namen. Standardmäßig handelt es sich hierbei um den Computernamen, auf dem der Application-Server (Tomcat) läuft. Er kann aber auch in der „web.xml“ oder „config.xml“ explizit festgelegt werden.

Sollen Skripte nur von der SAP-Schnittstelle ausgeführt werden, legt man sie in den Unterordner „ELO Archive Link“.

Unter diesen Ordnern können beliebig tiefe Strukturen erzeugt werden. Die Skripte in diesen Strukturen werden beim IndexServer-Start in einen gleich aufgebauten Dateisystemzweig kopiert. Somit können per @include Schlüsselwort Skripte andere „inkludieren“. Aber es bedeutet auch, dass in den Ordner- und Skriptnamen nur Zeichen verwendet werden dürfen, die im Dateisystem erlaubt sind.

Der IndexServer liest die Skripte nur beim Start ein. Wenn sich Skripte danach ändern, ist entweder ein Neustart nötig. Oder es wird der „Reload“-Button in der IndexServer-Optionen Seite geklickt. In der IndexServer API steht die Funktion reload für diesen Zweck zur Verfügung.

### 1.2.1 Aufruf per Namenskonvention

Wenn ein Ereignis ausgelöst wird, dann werden alle gleichnamigen Ereignisfunktionen aller Skripte aufgerufen. Die Reihenfolge ist dabei zufällig.

### 1.2.2 Exceptions werfen

Ereignisfunktionen können Exceptions werfen, z. B. um in onBeforeCheckinSord das Speichern des Objekts zu verhindern. Als Exception-Objekt wird entweder ein String oder ein Array bestehend aus einer Fehlernummer und einem Fehlertext weitergegeben:

```
throw “Only administrators are allowed to checkin objects of this type“;

throw [IXExceptionC.ACCESS_DENIED,
      “Only administrators are allowed to checkin objects of this type“];
```

### 1.2.3 Initialisierung

Die Toplevel-Objekte der Skripte werden beim IndexServer-Start ausgeführt.

### 1.2.4 Bisherige WF-Skripte

Das bisherige Konzept zur Ausführung von WF-Skripten bleibt unverändert.

## 1.3 JavaScript programmieren

### 1.3.1 Rhino JavaScript Engine

Der IndexServer benutzt die Rhino Script Engine von Mozilla zur Ausführung der Skripte, http://www.mozilla.org/rhino/ .

Zum Erlernen der Skriptsprache eignet sich z. B. „JavaScript The Definitive Guide“ von David Flanagan, O’Reilly, http://oreilly.com/catalog/9780596101992/.

### 1.3.2 Voraussetzungen

Neben dem Erlernen der einfachen JavaScript Programmiersprache sind gute Kenntnisse der IndexServer API erforderlich.

Dem Skriptprogrammierer sollte bewusst sein, dass fehlerhafte Skripte alle IndexServer-basierten Anwendungen stören können.

### 1.3.3 Beispiel

Das folgende Beispielskript startet einen Workflow, wenn ein neues Dokument mit einer bestimmten Ablagemaske eingecheckt wird.

```
// Import IndexServer API package
importPackage(Packages.de.elo.ix.client);

var maskName = "Invoice";
var templFlowId = "Invoice Workflow";

function onAfterCheckinSord(ec, sord, sordDB, parentSord, sordZ, lockZ) {

  // Parameter sordDB is undefined for new Sord objects?
  if (!sordDB) {

    // Is a document?
    if (sord.type >= SordC.LBT_DOCUMENT) {

      if (sord.maskName == maskName) {

        var wfId = ix.startWorkFlow(ec.ci,
                                    templFlowId,
                                    templFlowId + " - " + sord.name,
```

```
                                    sord.id);

        log.info("start workflow OK, wfId=" + wfId);
      }

    }
  }
}
```

### 1.3.4 Umfangreiches Beispiel

Im Verzeichnis examples finden Sie das Skript-Beispiel „ProtectArcLevels.js“, dass für Nicht-Administratoren die Änderung der obersten beiden Archivebenen verhindert. Zum Austesten kopieren Sie das Skript und das Verzeichnis „utils“ ins Archivverzeichnis „\Administration\IndexServer Scripting Base_ALL“. Anschließend laden Sie den IndexServer neu.

### 1.3.5 Tips

#### 1.3.5.1 java.io.File.delete() verwenden

In JavaScript ist „delete“ ein reserviertes Wort für den Lösch-Operator. Deshalb kann man ohne Weiteres die „delete“ Funktion der Java-Klasse „File“ nicht aufrufen.

Beispiel:

```
importPackage(Packages.java.io);

var f = new File(..);

//f.delete(); results in a syntax error
// solution:
f["delete"]();
```

# 2 WebService

Der IndexServer kann die Ereignisse via HTTP(S) zu einem WebService senden, der entweder in Java oder in einer .NET-Sprache implementiert ist. Der Datenaustausch erfolgt in einem proprietären, binären Übertragungsformat – nicht SOAP. Java-Programme benötigen dafür die EloixClient.jar, .NET-Programme die EloixClientCS.DLL.

## 2.1 Einrichten

Eine XML-Datei in einem Unterverzeichnis von „IndexServer Scripting Base“ beschreibt den WebService:

\Administration\IndexServer Scripting Base_ALL\myevents.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<ixserverevents url="http://localhost:1606/ixevents" />
```

Wie bei den Java-Skripten gilt auch hier, dass nur die Dateien im Unterverzeichnis „_ALL“ und in dem Unterverzeichnis, das den IX-Instanznamen trägt, ausgewertet werden. Sie müssen als Wurzelelement in der Datei „ixserverevents“ definieren. Aktuell wird als Attribut nur die URL zum WebService benötigt. Die XML-Definitionen im Unterverzeichnis „_ALL“ gelten auch für die „ELO Archive Link“ Anwendung.

## 2.2 Schnittstelle IXServerEvents

Der WebService muss die Schnittstelle IXServerEvents implementieren. Beim erstmaligen Aufruf einer Ereignisfunktion fragt der IndexServer über IXServerEvents.hasMethod(methodName) nach, ob eine Implementierung für das Ereignis bereitsteht. Gibt der WebService dafür „false“ zurück, dann wird das Ereignis nicht in diesem und auch nicht in den folgenden Fällen zum WebService weitergeleitet. Erst nach dem Neustart oder Neu-Laden des IndexServers wird hasMethod für jede Methode wieder aufgerufen.

OnBefore-Ereignisse sollten nur in Ausnahmefällen von einem WebService implementiert werden, weil andernfalls die Ausführungsgeschwindigkeit des IndexServers zu sehr leidet.

## 2.3 Beispiele

Ein Grundgerüst für einen Ereignis-WebService in C#.NET findet man im Paket IndexServer_Programming im Projekt .NET\source\ExamplesServerEventHandler. Dasselbe für die Java-Plattform stellt das Verzeichnis Java-1.5\source\EloixExampleServerEvents zur Verfügung.

# 3 Referenz

Die bereitgestellten Ereignisse sind in der Referenzdokumentation des IndexServers in der Klasse de.elo.ix.IXServerEvents beschrieben.