Kommentierter JavaScript Code

Version : 7.00.020

Im Normalfall muss man nicht in den generierten JavaScript Code eingreifen. Selbst wenn ein Fehler auftritt, z.B. weil eine Indexzeilenzuweisung einen Syntaxfehler aufweist, muss man nur selten den kompletten Code betrachten, da in der Statusanzeige die Fehlerstelle direkt angezeigt wird. Wenn jedoch ein komplexer Fehler auftritt oder die Laufzeitumgebung erweitert werden soll, ist es sinnvoll, dass man einen Überblick über die gesamte Konstruktion besitzt.

Inhalt

1 Ruleset Definition

In den folgenden Zeilen finden Sie den Ruleset, der dem generierten Code zugrunde liegt. Nicht alle Aktionen sind wirklich sinnvoll, manche Teile sind nur vorhanden um die komplette Bandbreite der Funktionen abzudecken.

```
<ruleset>
 <base>
  <name>Mailmaske2</name>

  <search>
   <name>"ELOOUTL2"</name>
   <value>"Thiele*"</value>
   <mask>2</mask>
   <max>200</max>
   <idate>
    <from>"-35"</from>
    <to>"+1"</to>
   </idate>
  </search>

  <masks>
    <mask>12</mask>
    <mask>13</mask>
    <mask>20</mask>
  </masks>

  <interval>1M</interval>
 </base>

 <rule>
  <name>Regel 1</name>
  <destination mask="5">"¶Ablage¶Mails¶" + ELOOUTL1</destination>
  <mask>20</mask>
  <index>
   <name>ADDENTRY</name>
   <value>getObjShort(2)</value>
  </index>
  <index>
   <name>ELOOUTL2</name>
```

```
 <value>"!!" + ELOOUTL2</value>
</index>
<index>
 <name>DOCDATE</name>
 <value>20070930</value>
</index>
<index>
 <name>ARCHIVINGMODE</name>
 <value>2</value>
</index>
<index>
 <name>ACL</name>
 <value>"PARENT"</value>
</index>
</rule>

<rule>
 <name>Journal-Kopie</name>
 <condition>ELOOUTL1.substring(0,3) != "ELO"</condition>
 <destination mask="1">"¶Ablage¶Journale¶" + ELOOUTL1</destination>
</rule>

<rule>
 <name>Script rule</name>
 <script>
  moveTo(Sord, "¶Ablage¶Ziele1¶" + ELOOUTL1);
  moveTo(Sord, "¶Ablage¶Ziele2¶" + ELOOUTL2);
  moveTo(Sord, "¶Ablage¶Ziele3¶" + ELOOUTL3);
 </script>
</rule>

<rule>
 <name>Global Error Rule</name>
 <condition>OnError</condition>
 <destination>"¶Ablage¶Error"</destination>
 <index>
  <name>ELOOUTL2</name>
  <value>"!!" + ELOOUTL2</value>
 </index>
 <index>
  <name>ARCHIVINGMODE</name>
  <value>0</value>
 </index>
```

```
 </rule>
</ruleset>
```

2 Generierter JavaScript Code

Die folgenden Zeilen enthalten den generierten Code zusammen mit Kommentaren zur Funktion oder zur Verwendung.

- Kommentare sind farblich und durch die Schriftformatierung hervorgehoben.
- Generierter Code wird im folgenden GELB markiert.
- HELLBLAU markierter Code kommt aus den mitgelieferten JavaScript Templates
- GRAU markierter Code kommt direkt aus der Anwenderdefinition der XML Datei.

Die folgenden Zeilen enthalten die Imports für die verwendeten externen Java Klassen und die Variablennamen für die Pseudo-Indexzeilen.

Wenn vom JavaScript Code aus direkt auf Java Klassen zugegriffen werden soll, dann müssen diese im Modul „Imports“ aufgeführt werden. Der Code aus diesem Modul wird immer ganz am Anfang des generierten Gesamtcodes plaziert.

```
//Import the IndexServer API classes.
importPackage(Packages.de.elo.ix.client);
importPackage(Packages.java.lang);
importPackage(Packages.java.sql);
importPackage(Packages.sun.jdbc.odbc);

var NAME;
var ARCDATE;
var DOCDATE;
var OBJCOLOR;
var OBJDESC;
var OBJTYPE;
var ARCHIVINGMODE;
var ACL;

var EM_PARENT_ID;
var EM_PARENT_ACL;
```

Die folgenden gelb markierten Zeilen enthalten generierten Code. Dieser wird aus der Definition der Suchmaske und der Liste der alternativen Masken erstellt und enthält die Variablendefinitionen für die Indexzeilen.

```
// Direct index line access variables.
// Duplicate group names are only defined at
// the first occurrence.

// From Mask EMail
var ELOOUTL1;
var ELOOUTL2;
var ELOOUTL3;
var ADDENTRY;
```

Da die Maske „Another Mail Mask“ genau die gleichen Indexzeilen wie die Maske „Email“ besitzt, werden hier keine weiteren Variablen angelegt. Diese wurde weiter oben bereits erzeugt.

```
// From Mask Another Mail Mask

// From Mask Barcode
var BARC;
var NUM;
var LIST;
var DTED;
var NUM0;
var NUM1;
var NUM2;
var NUM4;
var NUM6;
var USERLIST;
var EIX2;
var EIX3;
var SWLONLY;
var STATUS;
var ANW;
var KND;

// From Mask AchtIndex
var KDID;
var IX2;
var IX3;
var IX4;
var IX5;
var IX6;
```

```
var IX7;
var IX8;
var IXDAT;
var ISO;
var FIN;
var AUSWAHL;
```

Diese Zeilen enthalten ebenfalls generierten Code. Er wird aus der Definition des Suchbegriffs erzeugt.

```
// Compiled code from search definition
var EM_SEARCHNAME = "ELOOUTL2";
var EM_SEARCHVALUE = "Thiele*";
var EM_SEARCHCOUNT = 200;
var EM_SEARCHMASK = 2;
var EM_IDATEFROM = "-35";
var EM_IDATETO = "+1";
var EM_XDATEFROM = "";
var EM_XDATETO = "";
var EM_FOLDERMASK = "1";
```

Dieser Code wird aus den Listen des “Translate” Bereichs erzeugt.

```
var MonthNames = new Array();
MonthNames["01"] = "Januar";
MonthNames["02"] = "Februar";
MonthNames["03"] = "Maerz";
MonthNames["04"] = "April";
MonthNames["05"] = "Mai";
```

Die processRules Funktion wird aus den Regeln generiert. Für jede Fehlerregel wird ein try...catch Block erzeugt, der die zugehörenden normalen Regeln enthält.

```
function processRules(Sord) {
 try {
  processRule1(Sord);
  processRule2(Sord);
  processRule3(Sord);
 } catch (e) {
  log.info("Exception caught: " + e);
  processRule4(Sord);
  return;
 }
};
```

Für jede Regel wird eine Funktion mit dem Namen „processRule“ und der Nummer der Regel erzeugt.
Die erste Regel verwendet für neue Ordner die Masken-Id 5, wechselt auf die Dokumentenmaske 20, füllt die Indexzeilen ADDENTRY, ELOOUTL2 und DOCDATE mit neuen Werten und schaltet den Status auf Revisionssicher (ARCHIVNGMODE = 2) um. Als letztes wird ein neues Ablageziel mittels moveTo definiert.

```
function processRule1(Sord) {
  //Generated Rule code
  log.debug("Process Rule Regel 1.");
  EM_FOLDERMASK = 5;
  // Change Mask
  changeMask(Sord, 20);

  //Index line changes
  ADDENTRY = getObjShort(2);
  ELOOUTL2 = "!!" + ELOOUTL2;
  DOCDATE = 20070930;
  ARCHIVINGMODE = 2;
  ACL = "PARENT";
  //Compiled Code: destination
  moveTo(Sord, "¶Ablage¶Mails¶" + ELOOUTL1);
};
```

Die zweite Regel besitzt eine Bedingung, führt keine Änderungen an den Indexzeilen aus und definiert ebenfalls ein neues Ablageziel

```
function processRule2(Sord) {
  //Generated Rule code
  //Compiled Code: Condition
  if (ELOOUTL1.substring(0,3) != "ELO") {
    log.debug("Process Rule Journal-Kopie.");
    EM_FOLDERMASK = 1;
    //Index line changes
    //Compiled Code: destination
    moveTo(Sord, "¶Ablage¶Journale¶" + ELOOUTL1);
  }
};
```

Die dritte Regel ist eine JavaScript Regel. Dieser Code ist nicht generiert sondern wird direkt aus der Anwenderdefinition übernommen (Grau markiert). In diesem Beispiel werden direkt drei weitere Zielordner angegeben zu denen dann jeweils eine Referenz erzeugt wird.

```
function processRule3(Sord) {
```

```
  moveTo(Sord, "¶Ablage¶Ziele1¶" + ELOOUTL1);
  moveTo(Sord, "¶Ablage¶Ziele2¶" + ELOOUTL2);
  moveTo(Sord, "¶Ablage¶Ziele3¶" + ELOOUTL3);
};
```

Die letzte Regel ist eine Fehlerregel, das wird auch explizit im Kommentar vermerkt. Somit sind solche Regeln im JavaScript Code leicht zu erkennen. Sie löscht zuerst evtl. vorhandene neue Ablageziele, verändert eine Indexzeile und setzt den Status auf „Freie Bearbeitung“ zurück. Dieser wurde in der Regel 1 auf „Revisionssicher“ gesetzt, wenn die Fehlerregel das nicht zurücknehmen würde, könnte man später keine Korrekturen vornehmen.

```
function processRule4(Sord) {
 //Generated Error Rule, clear all old destinations
 EM_NEW_DESTINATION = new Array();
 log.info("Process Error Rule Global Error Rule.");
 EM_FOLDERMASK = 1;
 //Index line changes
 ELOOUTL2 = "!!" + ELOOUTL2;
 ARCHIVINGMODE = 0;
 //Compiled Code: destination
 moveTo(Sord, "¶Ablage¶Error");
};
```

Die letzte Fehlerregel wird zusätzlich unter dem Namen finalErrorRule zur Verfügung gestellt. Damit ist es möglich, diese Fehlerbehandlung auch von statischen oder vorkonfigurierten Code aus aufzurufen – die Nummer der Regel muss nicht bekannt sein.

```
function finalErrorRule(Sord) {
 processRule4(Sord);
}
```

Zur Suchmaske wird eine Funktion generiert die das gefundene SORD Objekt ausliest und die Indexzeilen in die vordefinierten Variablen überträgt. Die Basisdaten, wie z.B. das Ablagedatum oder die Farbe werden nicht durch generierten Code ausgelesen, da sie immer gleich sind. Sie werden später aus konfigurierten Code heraus erzeugt, dieser kann also im Projekt leicht angepasst oder erweitert werden.

```
function loadIndexLines(Sord) {
 ELOOUTL1 = getIndexValue(Sord, 0);
 ELOOUTL2 = getIndexValue(Sord, 1);
 ELOOUTL3 = getIndexValue(Sord, 2);
 ADDENTRY = getIndexValue(Sord, 3);
};
```

Zur Suchmaske gibt es ebenfalls eine generierte Funktion die die Indexzeilen wieder zurück in das SORD Objekt schreibt.

```
// Compiled code: write index lines of mask: EMail
function storeIndexLines2(Sord) {
 setIndexValue(Sord, 0,  ELOOUTL1);
 setIndexValue(Sord, 1,  ELOOUTL2);
 setIndexValue(Sord, 2,  ELOOUTL3);
 setIndexValue(Sord, 3,  ADDENTRY);
};
```

Für jede alternative Ablagemaske gibt es ebenfalls eine generierte Funktion die die Daten zurückschreibt. Es gibt keine Lesefunktion, da nur die Treffer der Suchmaske gelesen werden müssen.

```
// Compiled code: write index lines of mask: Another Mail Mask
function storeIndexLines20(Sord) {
 setIndexValue(Sord, 0,  ELOOUTL3);
 setIndexValue(Sord, 1,  ELOOUTL1);
 setIndexValue(Sord, 2,  ELOOUTL2);
 setIndexValue(Sord, 3,  ADDENTRY);
};


// Compiled code: write index lines of mask: Barcode
function storeIndexLines12(Sord) {
 setIndexValue(Sord, 0,  BARC);
 setIndexValue(Sord, 1,  NUM);
 setIndexValue(Sord, 2,  LIST);
 setIndexValue(Sord, 3,  DTED);
 setIndexValue(Sord, 4,  NUM0);
 setIndexValue(Sord, 5,  NUM1);
 setIndexValue(Sord, 6,  NUM2);
 setIndexValue(Sord, 7,  NUM4);
 setIndexValue(Sord, 8,  NUM6);
 setIndexValue(Sord, 9,  USERLIST);
 setIndexValue(Sord, 10,  EIX2);
 setIndexValue(Sord, 11,  EIX3);
 setIndexValue(Sord, 12,  SWLONLY);
 setIndexValue(Sord, 13,  STATUS);
 setIndexValue(Sord, 14,  ANW);
 setIndexValue(Sord, 15,  KND);
};
```

```
// Compiled code: write index lines of mask: AchtIndex
function storeIndexLines13(Sord) {
 setIndexValue(Sord, 0,  KDID);
 setIndexValue(Sord, 1,  IX2);
 setIndexValue(Sord, 2,  IX3);
 setIndexValue(Sord, 3,  IX4);
 setIndexValue(Sord, 4,  IX5);
 setIndexValue(Sord, 5,  IX6);
 setIndexValue(Sord, 6,  IX7);
 setIndexValue(Sord, 7,  IX8);
 setIndexValue(Sord, 8,  IXDAT);
 setIndexValue(Sord, 9,  ISO);
 setIndexValue(Sord, 10,  FIN);
 setIndexValue(Sord, 11,  AUSWAHL);
 setIndexValue(Sord, 12,  ANW);
 setIndexValue(Sord, 13,  KND);
};
```

Diese generierte Funktion entscheidet beim Speichern darüber, welche Speicherfunktion aufgerufen wird. Dazu liest sie die aktuelle Maskennummer aus, die durch die Regeln verändert sein kann und bestimmt die dazu passende Speicherfunktion. Falls eine unzulässige Maskennummer angegeben wurde, wird ein Laufzeitfehler ausgelöst.

```
function storeIndexLines(Sord) {
 var maskNo = Sord.getMask();
 if (maskNo == 2) {
  storeIndexLines2(Sord);
 } else if (maskNo == 20) {
  storeIndexLines20(Sord);
 } else if (maskNo == 12) {
  storeIndexLines12(Sord);
 } else if (maskNo == 13) {
  storeIndexLines13(Sord);
 } else {
  throw("Invalid mask id found, store aborted.");
 }
};
```

Der folgende Code kommt aus der Vorlage „Base Templates“, er kann also im Projekt an die jeweiligen Erfordernisse angepasst werden. Dabei sollte aber bedacht werden, dass dadurch ein Update erschwert wird. Deshalb sollten eigene Funktionen in ein

eigenes JavaScript Modul ausgelagert werden und von hier aus nur Zugriffe darauf statt finden.

```
//JavaScript Template: Base Templates
var Sords = new Array();
var EM_NEW_DESTINATION = new Array();
```

Die Funktion executeRuleset ist der zentrale Startpunkt der Verarbeitung. Sie wird vom Servlet im definierten Intervallabstand aufgerufen und führt die Suche und anschließende Abarbeitung der Treffermenge durch.

```
function executeRuleset(name, num){
 executeSearch();
 processResultSet();

 if (dbExit) {
  dbExit();
 }

 return "Idle...";
};
```

Für die Suche werden die im Ruleset definierten Suchparameter aus den Variablen EM_SEARCH... in eine Indexserversuche übertragen und die Suche ausgeführt. Die Treffermenge wird in dem Array Sords für die Weiterverarbeitung gespeichert.

```
function executeSearch() {
 log.info("Start Execute Search");
 ruleset.setStatusMessage("Searching...");
 var findInfo = new FindInfo();
 var findByIndex = new FindByIndex();

 var objKey = new ObjKey();
 var keyData = new Array(1);
 keyData[0] = EM_SEARCHVALUE;
 objKey.setName(EM_SEARCHNAME);
 objKey.setData(keyData);

 var objKeys = new Array(1);
 objKeys[0] = objKey;

 findByIndex.setObjKeys(objKeys);
 findByIndex.setMaskId(EM_SEARCHMASK);

 if ((EM_XDATEFROM != "") || (EM_XDATETO != "")) {
```

```
  var xdate = decodeDate(EM_XDATEFROM) + "..." + decodeDate(EM_XDATETO) ;
  findByIndex.setXDateIso(xdate);
  log.debug("Find by XDate: " + xdate);
 }

 if ((EM_IDATEFROM != "") || (EM_IDATETO != "")) {
  var idate = decodeDate(EM_IDATEFROM) + "..." + decodeDate(EM_IDATETO) ;
  findByIndex.setIDateIso(idate);
  log.debug("Find by IDate: " + idate);
 }

 findInfo.setFindByIndex(findByIndex);

 var findResult = ixConnect.ix().findFirstSords(findInfo, EM_SEARCHCOUNT, SordC.mbAll);

 Sords = findResult.getSords();
 ruleset.setMoreResults(findResult.isMoreResults());
 log.debug("More results available: " + findResult.isMoreResults());
 ixConnect.ix().findClose(findResult.getSearchId());
 log.info("Execute Search done, " + Sords.length + " entries found.");
 ruleset.setStatusMessage(Sords.length + " entries found");
}
```

Die Funktion processObject wird für jeden Eintrag aus der Trefferliste aufgerufen. Hier werden zuerst die Basisdaten (Kurzbezeichnung, Dokumentendatum etc.) und Indexzeilen in die JavaScript Indexvariablen übertragen und anschließend mit „processRules“ die Abarbeitung der Regeln durchgeführt. Danach werden die Daten in das Sord Objekt zurück übertragen und das Dokument verschoben und gespeichert.

```
function processObject(Sord) {
 loadBaseData(Sord);
 log.info("Sord: " + NAME);
 ruleset.setStatusMessage("Process: " + NAME);
 loadIndexLines(Sord)

 EM_NEW_DESTINATION = new Array();
 processRules(Sord);

 try {
  storeBaseData(Sord);
  storeIndexLines(Sord);
  moveFinally(Sord);
```

```
  ixConnect.ix().checkinSord(Sord, SordC.mbAll, LockC.NO);
  //moveFinally(Sord);
 } catch (e) {
  log.info("Error on store or move: "  + e);
  try {
   finalErrorRule(Sord);
   storeBaseData(Sord);
   storeIndexLines(Sord);
   ixConnect.ix().checkinSord(Sord, SordC.mbAll, LockC.NO);
   moveFinally(Sord);
  } catch (e) {
   log.info("Error in Error Rule: " + e);
  }
 }
}
```

Die Funktion „moveTo“ wird von dem <destination> Teil der Regel aufgerufen. An dieser Stelle wird aber noch nicht verschoben sondern nur die Ordnermaske und das Ziel zwischengespeichert. Die eigentliche Verschiebeoperation findet erst unmittelbar vor der Speicherung des Sord Objekts statt.

```
function moveTo(Sord, destination) {
 log.debug("MoveTo " + destination);
 EM_NEW_DESTINATION.push(EM_FOLDERMASK + "¶¶¶." + destination);
}
```

Die Funktion „moveFinally“ wird unmittelbar vor der Speicherung des Sord Objekts aufgerufen. Hier wird die Liste der <destination> Einträge abgearbeitet. Das Objekt wird in den Ordner des ersten Ziels verschoben, falls weitere Ziele eingetragen sind, werden entsprechende Referenzen angelegt. Wenn ein Ziel noch nicht existiert, wird es hier auch automatisch durch den Aufruf von „preparePath“ angelegt.

```
function moveFinally(Sord) {
 if (EM_NEW_DESTINATION.length > 0) {
  var destId = preparePath(EM_NEW_DESTINATION[0]);
  log.debug("Dest: " + destId + "   Source: " + Sord.getParentId());

  if ((destId > 0) && (destId != Sord.getParentId())) {
   ixConnect.ix().copySord(destId, Sord.getGuid(), null, CopySordC.MOVE);
   Sord.setParentId(destId);
  }

  if (ACL == "PARENT") {
```

```
  }

  var i;
  for (i = 1; i < EM_NEW_DESTINATION.length; i++) {
    var destId = preparePath(EM_NEW_DESTINATION[i]);
    log.debug("Add. Ref: Dest: " + destId + "   Source: " + Sord.getParentId());

    if ((destId > 0) && (destId != Sord.getParentId())) {
      ixConnect.ix().copySord(destId, Sord.getGuid(), null, CopySordC.REFERENCE);
    }
  }
 }
}
```

Die folgenden Programmteile stammen aus dem Modul ELO Utils. Hier sind verschiedene Hilfsroutinen abgelegt, die zur Verarbeitung benötigt werden.

Die Funktion „getIndexValue“ liest eine Indexzeile aus dem Sord Objekt aus und gibt den aktuellen Wert zurück. Spaltenindex Zeilen werden noch nicht unterstützt, können jedoch als Erweiterung hier implementiert werden.

```
//JavaScript Template: ELO Utils
function getIndexValue(Sord, lineNo) {
 var objKey = Sord.getObjKeys()[lineNo];
 if (!objKey) {
  return "";
 }

 var keyData = objKey.getData();
 if (keyData && keyData.length > 0) {
  return keyData[0];
 } else {
  return "";
 }
}
```

Mittels „setIndexValue“ wird ein Wert in eine Indexzeile zurück geschrieben. Spaltenindex Zeilen werden noch nicht unterstützt, können jedoch als Erweiterung hier implementiert werden.

```
function setIndexValue(Sord, lineNo, text) {
 var objKey = Sord.getObjKeys()[lineNo];
 var keyData = new Array(1);
 keyData[0] = text;
```

```
 objKey.setData(keyData);
}
```

Die Funktion „preparePath“ prüft nach, ob ein angeforderter Ablagepfad vorhanden ist. Fall ja, wird die ID des Zielregisters zurück gegeben. Wenn er noch nicht existiert, wird er unter Berücksichtigung der Ordnermaske angelegt.

```
function preparePath(destPath) {
 log.debug("PreparePath: " + destPath);
 var temp = destPath.split("¶¶¶.");
 if (temp.length == 2) {
  EM_FOLDERMASK = temp[0];
  destPath = temp[1];
 } else {
  EM_FOLDERMASK = "1";
 }

 try {
  var editInfo = ixConnect.ix().checkoutSord("ARCPATH:" + destPath, EditInfoC.mbOnlyId,
LockC.NO);
  log.debug("Path found, GUID: " + editInfo.getSord().getGuid() + "   ID: " +
editInfo.getSord().getId());
  EM_PARENT_ID = editInfo.getSord().getId();
  EM_PARENT_ACL = editInfo.getSord().getAclItems();
  return editInfo.getSord().getId();
 } catch (e) {
  log.debug("Path not found, create new: " + destPath + ", use foldermask: " +
EM_FOLDERMASK);
 }

 EM_PARENT_ID = -1;

 items = destPath.split("¶");

 var sordList = new Array(items.length - 1);

 var i;
 for (i = 1; i < items.length; i++) {
  log.debug("Split " + i + " : " + items[i]);
  var editInfo = ixConnect.ix().createSord("1", EM_FOLDERMASK,  EditInfoC.mbSord);
  var sord = editInfo.getSord();
  sord.setName(items[i]);
  sordList[i - 1] = sord;
 }
```

```
 log.debug("now checkinSordPath");
 var ids = ixConnect.ix().checkinSordPath("1", sordList,  SordC.mbMin);
 log.debug("checkin done: id: " + ids[ids.length - 1]);

 return ids[ids.length - 1];
}
```

Die Funktion „loadBaseData“ wird vor der Verarbeitung jedes Sord Objekts aufgerufen. Sie überträgt die Basisdaten in die JavaScript Indexvariablen. Wenn weitere Werte zur Verfügung gestellt werden sollen, dann muss zuerst in Imports die zusätzliche Indexvariable definiert werden und anschließend in load/storeBaseData die Übertragung hinzu gefügt werden.

```
function loadBaseData(Sord) {
 NAME = Sord.getName();
 DOCDATE = Sord.getXDateIso();
 ABLDATE = Sord.getIDateIso();
 OBJCOLOR = Sord.getKind();
 OBJDESC = Sord.getDesc();
 OBJTYPE = Sord.getType();
 ARCHIVINGMODE = Sord.getDetails().getArchivingMode() - 2000;
 ACL = "";
}
```

Die Funktion „storeBaseData“ wird nach der Verarbeitung jedes Sord Objekts vor dem Speicher aufgerufen. Sie überträgt den Inhalt der JavaScript Indexvariablen zurück in das Sord Objekt.

```
function storeBaseData(Sord) {
 if (NAME != "") {
  Sord.setName(NAME);
 }
 Sord.setXDateIso(DOCDATE);
 Sord.setIDateIso(ABLDATE);
 Sord.setKind(OBJCOLOR);
 Sord.setDesc(OBJDESC);
 Sord.setType(OBJTYPE);
 Sord.getDetails().setArchivingMode(ARCHIVINGMODE + 2000);
 processAcl(Sord);
}
```

Eine Veränderung der ACL erlaubt im Augenblick nur einen Wert: PARENT. Hierüber wird definiert, dass das Objekt beim Speichern die Berechtigungseinstellung des neuen Ablageziels übernimmt.

```
function processAcl(Sord) {
 if (ACL == "PARENT") {
  var aclItems = new Array(1);
  var parentAcl = new AclItem(0, 0, "", AclItemC.TYPE_INHERIT);
  aclItems[0] = parentAcl;
  Sord.setAclItems(aclItems);
 }
}
```

Die Hilfsfunktion „processResultSet“ läuft über die gesamten Trefferliste und ruft für jeden Eintrag die Verarbeitungsfunktion auf.

```
function processResultSet() {
 var i;
 for (i = 0; i < Sords.length; i++) {
  processObject(Sords[i]);
 }

 ruleset.setStatusMessage("Wait.");
}
```

Mittels der Funktion „changeMask“ kann die Ablagemaske des aktuellen Sord Objekts geändert werden. Da der Inhalt der Indexzeilen in den JavaScript Variablen liegt, müssen hier keine Daten kopiert werden. Lediglich die leere ObjKeys Struktur der neuen Maske wird in das aktuelle Objekt übernommen.

```
function changeMask(Sord, newMaskId) {
 log.debug("Switch to new MaskId: " + newMaskId);
 var editInfo = ixConnect.ix().changeSordMask(Sord, newMaskId, EditInfoC.mbSord);
 Sord.setMask(editInfo.getSord().getMask());
 Sord.setObjKeys(editInfo.getSord().getObjKeys());
}
```

Die Funktion „pad“ gibt eine Zahl als String fester Länge zurück. Dieser ist bei Bedarf mit führenden Nullen aufgefüllt, z.B. um Monate zu Formatieren: März = 03.

```
function pad(val, len) {
 val = String(val);
 while (val.length < len) val = "0" + val;
 return val;
}
```

Die Funktion „isoDate“ gibt einen formatierten Datumsstring YYYYMMDD aus dem Datumsparameter zurück.

```
function isoDate(date) {
 return pad(date.getFullYear(), 4) + pad(date.getMonth() + 1, 2) + pad(date.getDate(), 2);
}
```

Die Funktion „decodeDate“ erwartet als Eingabe entweder ein Datum im ISO Format oder eine positive oder negative Zahl. Das Datum wird direkt zurück gegeben, die Zahlen werden als Offset zum aktuellen Tagesdatum gewertet und ein entsprechend berechnetes Datum zurück gegeben. Diese Funktion wird für die Datumseinschränkung bei der Suche verwendet.

```
function decodeDate(text) {
 if (text == "") {
  return text;
 }
 if (text.charAt(0) == '+') {
  text = text.substring(1);
  var now = new Date();

  var dateOffset = (24*60*60*1000) * text;
  now.setTime(now.getTime() + dateOffset);

  return isoDate(now);
 } else  if (text.charAt(0) == '-') {
  text = text.substring(1);
  var now = new Date();

  var dateOffset = 0 - ( (24*60*60*1000) * text);
  now.setTime(now.getTime() + dateOffset);

  return isoDate(now);
 } else {
  return text;
 }
}
```

Die folgenden Routinen implementieren einen experimentellen Zugriff auf externe Datenbanken. Sie sind nicht Bestandteil der eigentlichen Kernimplementierung sondern demonstrieren die Erweiterbarkeit des Ansatzes. Für die externe Verwendung wird eine Funktion „getObjShort“ zur Verfügung gestellt. Sie liest aus einer ELOoffice Archivdatenbank zu einer gegebenen Objekt-Id die zugehörende Kurzbezeichnung aus liefert den Text zurück. Vor der Ausführung wird geprüft, ob die

Datenbankverbindung bereits initialisiert ist und bei Bedarf wird eine Verbindung hergestellt. Am Ende der Verarbeitung wird diese dann wieder getrennt.

```
//JavaScript Template: DB Access
var dbinitdone;
var dbcn;
```

Prüft, ob die Datenbankverbindung bereits existiert und stellt sie bei Bedarf her. In diesem einfachen Beispiel wird eine Datenbank mit einem festen Namen „EMDemo.mdb“ aus dem Verzeichnis „c:\temp“ verwendet.

```
function dbInit() {
 if (dbinitdone == true) {
  return;
 }

 log.debug("Now init ODBC driver");

 try {
  var treiber = "sun.jdbc.odbc.JdbcOdbcDriver";
  Class.forName(treiber).newInstance();

  log.debug("Register driver ODBC");
  DriverManager.registerDriver(new JdbcOdbcDriver());

  log.debug("Get Connection");
  var dbUrl = "jdbc:odbc:Driver={Microsoft Access Driver
(*.mdb)};DBQ=C:\\Temp\\EMDemo.mdb";
  dbcn = DriverManager.getConnection(dbUrl,"","");

  log.debug("Init done.");
 } catch (e) {
  log.debug("ODBC Exception: " + e);
 }

 dbinitdone = true;
}
```

Die „dbExit“ Funktion wird am Ende der Verarbeitung aufgerufen und kann dazu verwendet werden, die Verbindung wieder zu trennen. In diesem Fall muss die Variable dbinitdone wieder auf false zurück gesetzt werden, damit bei der nächsten Verarbeitung eine neue Verbindung geöffnet wird. In diesem einfachen Beispiel bleibt eine einmal geöffnete Datenbankverbindung dauerhaft offen.

```
function dbExit() {
```

```
  log.debug("dbExit");
}
```

Die Funktion „getObjShort“ stellt die eigentliche Funktionalität des Datenbankzugriffs zur Verfügung. Sie kann in den Zuweisungsausdrücken der Indexzeilen in den Regeln verwendet werden: OBJSHORT = getObjShort(2);

```
function getObjShort(objId) {
  log.debug("getObjShort: " + objId);

  dbInit();

  log.debug("createStatement");
  var p = dbcn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_UPDATABLE);

  log.debug("executeQuery");
  var qry = "SELECT objshort FROM objekte where objid =" + objId + ";";
  var rss = p.executeQuery(qry);
  var res = "Not found";

  log.debug("read result");
  if (rss.next()) {
    log.debug("Result: " + rss.getString(1));
    res = rss.getString(1);
  }

  rss.close();
  p.close();

  return res;
}
```