Mailpostfach überwachen

JavaScript Library Version 7.00.032

Seit der Version 7.00.032 enthält die ELOas JavaScript Library eine Modul zum Senden und Empfangen vom Emails. Diese Anleitung soll zeigen, wie man ELOas zur Überwachung eines Postfachs verwenden kann.

Hinweis: dieses Beispiel soll keine Mail Archivierung simulieren. Dafür haben wir andere Module in unserer Produktliste, die dafür geeignet sind. Es soll vielmehr als Basis für „Autoresponder“ dienen, also Programme die auf eine Mail automatisch eine Aktion auslösen (z.B. ein Anwender sendet eine Registrierungsmail und daraufhin wird sein Konto freigeschaltet).

1 Allgemeine Vorgehensweise

Bevor ein Ruleset zur Bearbeitung von Postfächern erstellt werden kann, muss im Modul „mail“ eine Postfachverbindung erstellt werden. Da es hier extrem viele Unterschiede und Möglichkeiten gibt, kann man hier nicht mit einer einfachen Konfigurationsliste arbeiten. Stattdessen ist es notwendig, dass für jede Postfachverbindung eine „connect“ Methode erstellt wird. Diese muss die Verbindung zum Mailserver aufbauen, das richtige Postfach auswählen und die Nachrichtenliste einlesen.

Jede Postfachverbindung erhält einen einfachen, kurzen Namen – z.B. GMAIL. Dieser Name wird an verschiedenen Stellen benötigt und muss „Identifier-Tauglich“ sein, d.h. er muss mit einem Buchstaben beginnen und darf danach weitere Buchstaben oder Ziffern enthalten (aber keine Umlaute, das sind aus amerikanischer Sicht keine Buchstaben). Dieser Name wird dann an verschiedenen Stellen im Ruleset und in der JavaScript Implementierung benötigt.

1.1 Verbindungsaufbau

Die JavaScript Library bringt bereits in der Standardinstallation eine Definition für eine Verbindung mit dem Namen GMAIL mit. Diese werden wir für das Beispiel verwenden. Da der Verbindungsname in die speziellen Funktionen einfließt, können Sie auch mehrere Verbindungen parallel definieren und in unterschiedlichen Rulesets verwenden.

Die Standardfunktion für den GMAIL Verbindungsaufbau sieht so aus:

```
connectImap_GMAIL: function() {
  var props = new Properties();
```

```
  props.setProperty("mail.imap.host", "imap.gmail.com");
  props.setProperty("mail.imap.port", "993");
  props.setProperty("mail.imap.connectiontimeout", "5000");
  props.setProperty("mail.imap.timeout", "5000");
  props.setProperty("mail.imap.socketFactory.class", "javax.net.ssl.SSLSocketFactory");
  props.setProperty("mail.imap.socketFactory.fallback", "false");
  props.setProperty("mail.store.protocol", "imaps");

  var session = Session.getDefaultInstance(props);
  MAIL_STORE = session.getStore("imaps");
  MAIL_STORE.connect("imap.gmail.com", "<ANWENDER>@gmail.com", "<PASSWORT>");
  var folder = MAIL_STORE.getDefaultFolder();
  MAIL_INBOX = folder.getFolder("INBOX");
  MAIL_INBOX.open(Folder.READ_WRITE);
  MAIL_MESSAGES = MAIL_INBOX.getMessages();
  MAIL_DELETE_ARCHIVED = false;
},
```

Das Beispiel verbindet sich mit dem GOOGLEMAIL Server „imap.gmail.com“ auf dem Port „993“ über eine verschlüsselte Verbindung („mail.store.protocol“ – „imaps“). Diese Informationen werden in ein Property Objekt eingetragen. Ihr eigener Mail Server benötigt evtl. andere Werte – diese müssen Sie der Mailserver-Anleitung entnehmen.

Hinweis: wenn Sie ein Google Mail Konto einrichten, müssen Sie es zuerst für einen IMAP Zugriff freischalten. Das ist möglich unter „Einstellungen“ – „Weiterleitung und POP/IMAP“ – „IMAP aktivieren“.

Die Anmeldung wird dann über den Befehl MAIL_STORE.connect ausgeführt. Hier muss nochmals der Servername angegeben werden und der Postfachanwender mit Passwort.

Nach der Anmeldung wird als nächstes der „Posteingang“ Ordner aufgesucht. Man könnte durchaus auch andere Ordner überwachen, z.B. „Gesendet“:

```
  MAIL_INBOX = folder.getFolder("[Google Mail]/Gesendet");
```

Über den Befehl MAIL_INBOX.getMessages() werden schließlich alle Mails des Ordners eingelesen und in die interne Message Liste aufgenommen. Diese Liste wird dann später abgearbeitet indem der Ruleset für jeden Eintrag dieser Liste einmal aufgerufen wird.

Die Variable „MAIL_DELETE_ARCHIVED“ bestimmt, ob der Ruleset nach erfolgreicher Bearbeitung die Nachricht löschen oder als bearbeitet markieren darf. Wenn er, wie in der Voreinstellung, auf false steht, wird der Nachrichtenstatus nicht verändert. Das ist besonders in der Testphase sehr praktisch, damit man sich nicht ständig neue Mails erzeugen muss. Im Betrieb wird dieser Eintrag im Normalfall auf true stehen.

1.2 Ruleset erstellen

Ein einfacher Ruleset zur Abarbeitung des Postfachinhalts besteht aus zwei wesentlichen Teilen: der Definition der Suche und dem Skript zur Abarbeitung der Mail.

Die Suche wird folgendermaßen definiert:

```
<search>
<name>"MAILBOX_GMAIL"</name>
<value>"ARCPATH:¶IMAP"</value>
<mask>2</mask>
<max>200</max>
</search>
```

Der Suchname „MAILBOX_GMAIL“ signalisiert, dass es sich nicht um eine normale Archivsuche sondern um ein Postfach mit dem Verbindungsnamen GMAIL handelt. Die erzeugten ELO Dokumente werden im Archivschrank „IMAP“ abgelegt (über „ARCPATH:¶IMAP“) und werden mit der Maske 2 (Email in einem Standard ELO Archiv) erzeugt. Die Zahl der Treffer wird im Normalfall nicht weiter beachtet, sie sollte trotzdem eingetragen werden damit es im Designer nicht zu einer Fehlermeldung kommt.

Das Skript zur Ausführung wird im Wesentlichen von der benötigten Funktion bestimmt. Ein ganz einfacher Rahmen könnte so aussehen:

```
<script>
  log.debug("Process Mailbox: " + NAME);
  OBJDESC = mail.getBodyText(MAIL_MESSAGE);
  ELOOUTL1 = mail.getSender(MAIL_MESSAGE);
  ELOOUTL2 = mail.getRecipients(MAIL_MESSAGE, "¶");
  EM_WRITE_CHANGED = true;
  MAIL_ALLOW_DELETE = true;
</script>
```

Bei der Skriptausführung ist die Nachricht in der Variablen MAIL_MESSAGE verfügbar. Hieraus können dann Standardwerte wie Mailtext, Absender und Empfänger ausgelesen werden. Damit das etwas einfacher geht, stellt das Modul mail hierfür die Hilfsroutinen getBodyText, getSender und getRecipients zur Verfügung.

Der Betreff wird automatisch in die Kurzbezeichnung (NAME) übernommen. Der Mailkörper wird in den Zusatztext geschrieben und Absender und Empfänger in die jeweiligen Indexzeilen übertragen. Zuletzt wird die Nachricht über MAIL_ALLOW_DELETE als Bearbeitet markiert oder gelöscht.

Das komplette Beispiel sieht dann so aus:

```
<ruleset>
```

```
<base>
<name>Mailbox</name>
<search>
<name>"MAILBOX_GMAIL"</name>
<value>"ARCPATH:¶IMAP"</value>
<mask>2</mask>
<max>200</max>
</search>
<interval>10M</interval>
</base>
<rule>
<name>List</name>
<condition></condition>
<script>
 log.debug("Process Mailbox: " + NAME);
 OBJDESC = mail.getBodyText(MAIL_MESSAGE);
 ELOOUTL1 = mail.getSender(MAIL_MESSAGE);
 ELOOUTL2 = mail.getRecipients(MAIL_MESSAGE, "¶");
 EM_WRITE_CHANGED = true;
 MAIL_ALLOW_DELETE = true;
</script>
</rule>
<rule>
<name>Global Error Rule</name>
<condition>OnError</condition>
<script></script>
</rule>
</ruleset>
```

2 Überwachte Bearbeitung

Das einfache Beispiel hat einen wesentlichen Nachteil: wenn eine Mail bereits als „Bearbeitet“ markiert oder gelöscht wurde und der Prozess abgebrochen wird, bevor die Daten im Archiv gespeichert werden konnten, bleibt ein Datensatz unbearbeitet. Dieses Problem kann man komplett vermeiden, indem man mit einem zweistufigen Ansatz arbeitet: eine neue Mail wird zuerst nur im ELO gespeichert aber noch nicht gelöscht. Erst wenn bei einem späteren Lauf eine Mail gefunden wird, die bereits im ELO vorhanden ist, wird sie gelöscht.

Für diese Vorgehensweise sind zwei Voraussetzungen nötig: die Mail muss eindeutig erkennbar sein und bei der Abarbeitung muss geprüft werden, ob sie bereits im Archiv vorhanden ist. Der erste Punkt ist leicht erfüllbar: jede Mail hat eine interne Mail-ID. Diese kann im ELO in einer Indexzeile gespeichert werden (z.B. in der Standard Mail Maske in der

Indexzeile ELOOUTL3, diese ist für die Mail-Id vorgesehen). Der zweite Punkt kann durch eine Hilfsroutine aus dem Modul ix leicht erfüllt werden: ix.lookupIndexByLine.

Das geänderte Skript sieht dann so aus:

```
<script>
 log.debug("Process Mailbox: " + NAME);
 // wenn die Nachricht bereits im Archiv ist: dann löschen.
 var msgId = MAIL_MESSAGE.messageID;
 if (ix.lookupIndexByLine(EM_SEARCHMASK, "ELOOUTL3", msgId) != 0) {
  log.debug("Mail bereits im Archiv vorhanden, Ignorieren oder Löschen");
  MAIL_ALLOW_DELETE = true;
 } else {
  OBJDESC = mail.getBodyText(MAIL_MESSAGE);
  ELOOUTL1 = mail.getSender(MAIL_MESSAGE);
  ELOOUTL2 = mail.getRecipients(MAIL_MESSAGE, "¶");
  ELOOUTL3 = msgId;
  EM_WRITE_CHANGED = true;
 }
</script>
```

3 Markieren statt löschen

In der Standardimplementierung wird eine bearbeitete Mail aus dem Postfach gelöscht. Das ist in manchen Fällen nicht gewünscht. Stattdessen kann man auch eine Markierung vornehmen. Ein möglicher Kandidat ist das „Gelesen“ Flag. Eine bearbeitete Mail wird vom ELOas auf gelesen gesetzt und unterscheidet sich damit von einer neuen Mail. In diesem speziellen Fall müssen neben der connectImap Methode noch weitere Methoden in der mail JavaScript Library definiert werden:

nextImap_GMAIL(): diese Funktion schaltet auf die nächste Nachricht weiter. Sie muss in diesem Beispiel prüfen, ob eine Mail bereits als gelesen markiert wurde und diese dann bei Bedarf überspringen.

finalizeImap_GMAIL(): diese Funktion markiert die bearbeitete Nachricht. In der Standardimplementation wird die Nachricht gelöscht. In unserem Beispiel hingegen soll sie nur als Gelesen markiert werden.

3.1 nextImap_GMAIL

Diese Funktion schaltet auf die nächste Nachricht weiter. Dafür läuft sie sequenziell über die Nachrichtenliste, die aktuelle Position wird in der Variablen MAIL_POINTER gespeichert. Wenn eine Nachricht bereits als gelesen markiert wurde, wird sie übersprungen. Bei der ersten ungelesenen Nachricht wird diese aktiviert (sprich – in die Variable MAIL_MESSAGE kopiert) und der Wert true zurück geliefert. Wenn es keine weiteren Nachrichten gibt, wird ein false

zurückgeliefert. Der ELOas beendet dann die Abarbeitung dieses Rulesets und wechselt zum Nächsten.

```
nextImap_GMAIL: function() {
 for (;;) {
  if (MAIL_POINTER >= MAIL_MESSAGES.length) {
   return false;
  }

  MAIL_MESSAGE = MAIL_MESSAGES[MAIL_POINTER];

  var flags = MAIL_MESSAGE.getFlags();
  if (flags.contains(Flags.Flag.SEEN)) {
   MAIL_POINTER++;
   continue;
  }

  MAIL_ALLOW_DELETE = false;
  MAIL_POINTER++;
  return true;
 }

 return false;
},
```

Neben der Weiterschaltung wird noch eine Initialisierung vorgenommen: die Variable MAIL_ALLOW_DELETE wird auf false gesetzt. Nur dann, wenn im Ruleset eine Bearbeitung vorgenommen wurde, sollte dieser die Variable auf true setzen. In diesem Fall wird die Mail dann in der finalizeImap Methode als bearbeitet markiert.

3.2 finalizeImap_GMAIL

Die Funktion finalizeImap_GMAIL muss eine Mail als Bearbeitet markieren, das passiert durch setzen des Flags „SEEN“. Es darf aber nur denn gesetzt werden, wenn die Connect Methode es prinzipiell erlaubt (MAIL_DELETE_ARCHIVED) und der Ruleset die aktuelle Mail als archiviert markiert hat (MAIL_ALLOW_DELETE).

```
finalizeImap_GMAIL: function() {
 if (MAIL_DELETE_ARCHIVED && MAIL_ALLOW_DELETE) {
  message.setFlag(Flags.Flag.SEEN, true);
 }
},
```