Automatische Skriptinstallation

[Stand: 25.03.2019 | Programmversion: 12.00.000]

Seit der Version 9.01 wird eine neue Funktion zur automatischen Skriptinstallation unterstützt. Diese wurde in der Version 9.03 noch mal erweitert. Die Skripte müssen dazu in ein Installationspaket gepackt werden, welche aus einem ZIP-Archiv bestehen, das das Installationsskript und bei Bedarf weitere Nutzerdaten enthält. Das Paket kann entweder per Drag-and-drop oder über einen Download-Link übergeben werden.

Inhalt

1 Benutzersicht auf die neue Funktion 2
2 Aufbau eines Installationspakets 3
3 Einfaches Installationsskript 4
4 Installation mit User Interface 6
4.1 Verwendung von Callback-Aufrufen 11
4.2 Einfügen von Library-Dateien 11
4.3 Einfügen von Konfigurationsdateien 11
5 Installation von einer Webseite 13
5.1 Ermittlung der Base64-URL 15

1 Benutzersicht auf die neue Funktion

Die Installationspakete können entweder als ZIP-Archiv mit der Extension *.eloinst* angeboten werden oder über den elodms-Protokoll-Handler als Download-Link in eine Webseite eingebettet werden.

Wenn der Benutzer so ein Paket per Drag-and-drop in das Archiv zieht bzw. den Link anklickt, wird der Installationsvorgang gestartet. Der Benutzer muss als Administrator angemeldet sein und ausreichende Rechte in der Archivstruktur besitzen.

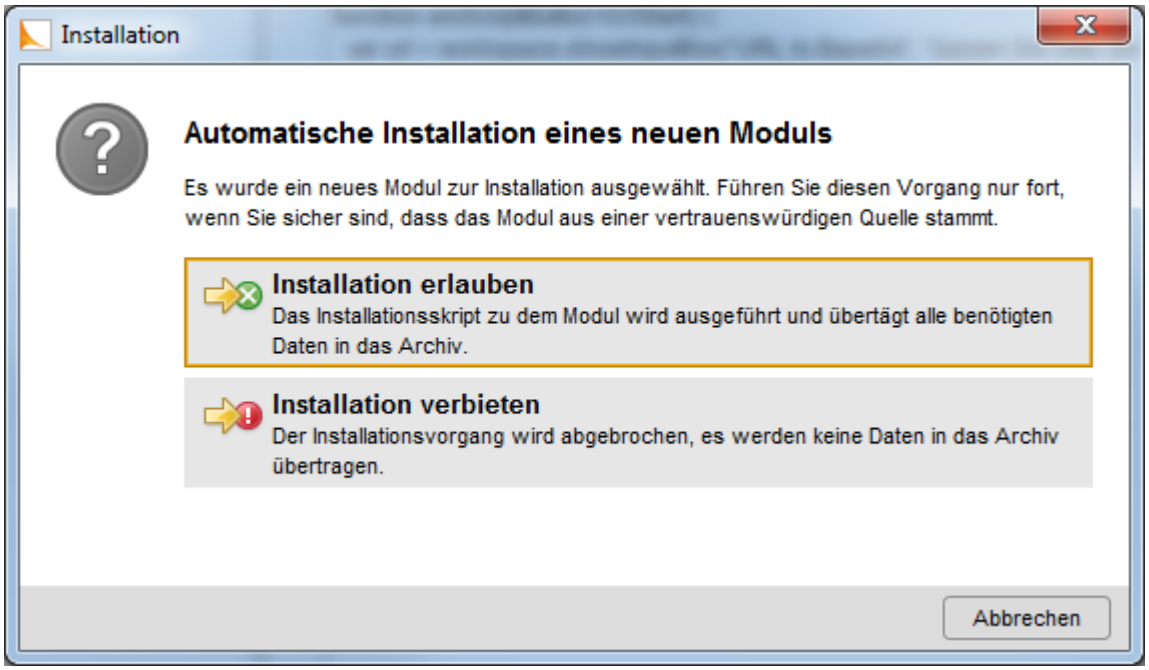


Abb. 1: Schaltfläche 'Installation erlauben'

Ein Installationsvorgang muss vom Benutzer immer ausdrücklich erlaubt werden. Eine stille Installation im Hintergrund ist aus Sicherheitsgründen nicht vorgesehen. Sobald der Administrator die Installation freigegeben hat, wird die Datei heruntergeladen und entpackt. Anschließend wird das Installationsskript ausgeführt.

2 Aufbau eines Installationspakets

Ein Installationspaket besteht aus zwei Bereichen:

1. Dem Installationsskript – es trägt immer den Namen *install.js* und kann beliebige Skriptkommandos für den ELO Java Client enthalten.
2. Dem Nutzdatenverzeichnis – es trägt den Namen *install.data* und kann beliebige Unterstrukturen enthalten.

Diese beiden Teile werden in ein ZIP-Archiv zusammengefasst und mit der Extension *.eloinst* gespeichert.

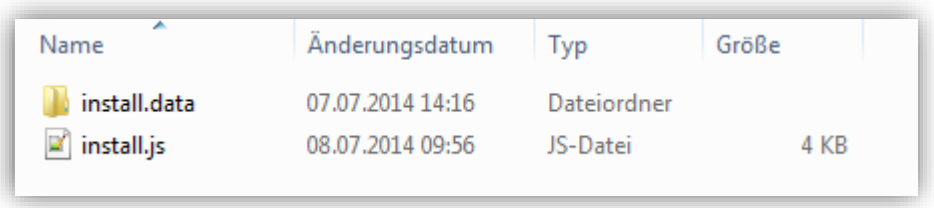


Abb. 2: Nutzdatenverzeichnis

Das Nutzdatenverzeichnis wird im Allgemeinen einen oder mehrere Importdatensätze enthalten. Diese können vom Skript aufgegriffen werden und über das Skriptkommando `workspace.doImport` in das Archiv übertragen werden.

Alternativ dazu können aber auch beliebige Dateien hinterlegt werden, welche dann vom Skript manuell in das Archiv eingetragen werden.

3 Einfaches Installationsskript

Im Folgenden soll ein einfaches Installationsskript vorgestellt werden. Es handelt sich um die Standardfunktion *GotoId* – ein Skript welches über eine Eingabe eine ELO Objekt-Id oder GUID abfragt und anschließend an diese Archivposition springt. Das Standardskript besteht aus der Skriptdatei *GotoId.js* und zusätzlichen Textdateien zur Internationalisierung der enthaltenen Texte. Diese Dateien wurden exportiert und liegen als Datei *GotoId.zip* im Verzeichnis *install.data* vor.

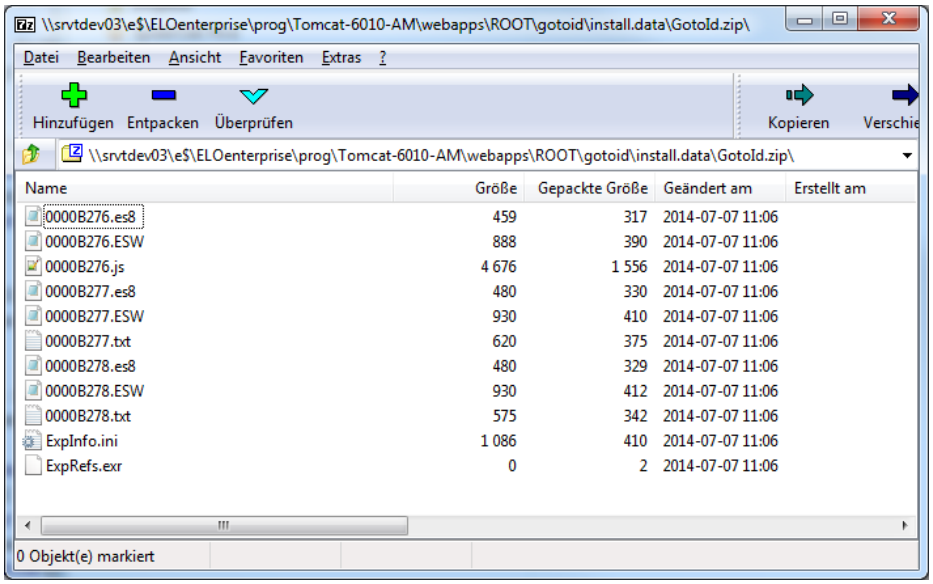


Abb. 3: Inhalt der Datei 'GotoId.zip'

Das Installationsskript muss als erstes das Verzeichnis mit den Nutzdaten ermitteln. Dafür gibt es im Objekt *directories* das neue Property *payloadDir*:

```
var payloadDir = workspace.directories.payloadDir.path;
```

Als nächstes muss das Zielverzeichnis ermittelt werden. Das ist bei einem Skript für den ELO Java Client das Skriptverzeichnis unter *Administration*. Dieses Verzeichnis besitzt eine spezielle GUID und kann hierüber auch dann ermittelt werden, wenn es vom Administrator umbenannt wurde (auch wenn man den Namen im Normalfall nicht ändern sollte). Die GUID `(E10E1000-E100-E100-E100-E10E10E10E11)` kann man sich aus dem Verschlagwortungsdialog herauskopieren.



Abb. 4: ID und GUID in der Verschlagwortung

```
var jcScriptFolder = archive.getElementByGuid('(E10E1000-E100-E100-E100-
E10E10E10E11)');
```

Nachdem der Zielordner ermittelt wurde, kann der Importvorgang gestartet werden. Die Methode *doImport* erwartet als Parameter den Namen der Importdatei, das Zielverzeichnis und weitere Daten zum Importvorgang.

```
workspace.doImport(payloadDir + "\\GotoId.zip", jcScriptFolder.id,
ImportOptionsC.GUIDS_KEEP, 0);
```

Hier nochmal das komplette Skript:

```
var payloadDir = workspace.directories.payloadDir.path;

var jcScriptFolder = archive.getElementByGuid('(E10E1000-E100-E100-E100-
E10E10E10E11)');
workspace.doImport(payloadDir + "\\GotoId.zip", jcScriptFolder.id,
ImportOptionsC.GUIDS_KEEP, 0);
```

Bei so einfachen Skripten lohnt es sich noch nicht, diese in Teilfunktionen zu strukturieren. Bei umfangreicheren Installationen kann es aber durchaus notwendig werden. Aus diesem Grund sollte man prinzipiell einen Funktionsaufruf einsetzen.

Zudem sollte man auch eine Fehlerbehandlung einplanen, die dem Benutzer sprechende Fehlermeldungen bietet, falls eine Installation nicht möglich ist.

```
doInstall();

function doInstall() {
  var payloadDir = workspace.directories.payloadDir.path;

  try {
    var jcScriptFolder = archive.getElementByGuid('(E10E1000-E100-E100-
E100-E10E10E10E11)');
  } catch(e) {
    workspace.showAlertBox("ELO Install", "Cannot find Java Client
Scripting Folder, Import aborted");
    return;
  }

  workspace.doImport(payloadDir + "\\GotoId.zip", jcScriptFolder.id,
ImportOptionsC.GUIDS_KEEP, 0);
}
```

4 Installation mit User Interface

Da dem Installationsskript alle Skriptaufrufe des ELO Java Clients zur Verfügung stehen, kann man auch interaktiv mit dem Benutzer arbeiten. Es besteht also die Möglichkeit, während der Installation z.B. Servernamen abzufragen und in Konfigurationsdateien einzutragen.

Als Beispiel soll die ELOas Funktion zur Benachrichtigung von Workflowaufgaben dienen. Hier sind Dateien im ELOas Bereich und im ELO Java Client-Skript-Bereich zu installieren. Zudem müssen auch noch Benutzerspezifische Daten zum Mail-Server und -Adressen abgefragt und eingetragen werden.

Das Installationsskript *Notify* prüft ab, ob es bereits eine Rule mit den Namen *NotifyWf* gibt. Wenn nicht, werden die benötigten Parameter abgefragt und aus einem Template die Rule-Datei erstellt und im Archiv abgelegt.

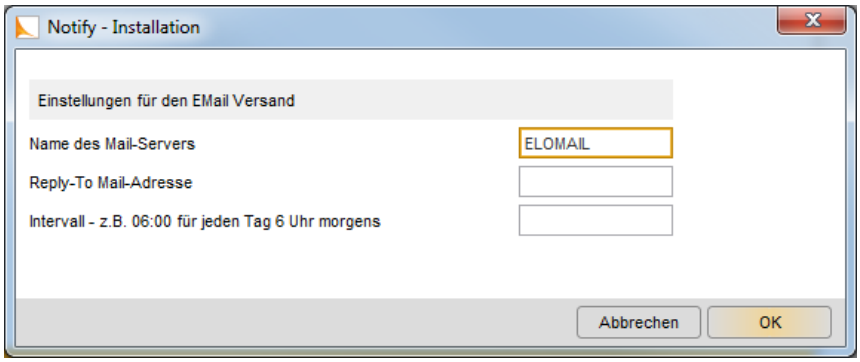


Abb. 5: Beispielsinstallation

```
function queryParams() {
  var dlg = workspace.createGridDialog("Notify - Installation", 3, 6);
  var panel = dlg.gridPanel;

  var labelText = '<html><div style="width:400px; padding:5px; background-
color: #f0f0f0">' +
        'Einstellungen für den EMail Versand' +
        '</div></html>';

  panel.addLabel(1,2,2, labelText);

  panel.addLabel(1,3,1, "Name des Mail-Servers");
  var serverName = panel.addTextField(2,3,1);

  panel.addLabel(1,4,1, "Reply-To Mail-Adresse");
  var replyTo = panel.addTextField(2,4,1);

  panel.addLabel(1,5,1, "Intervall - z.B. 06:00 für jeden Tag 6 Uhr
morgens");
  var interval = panel.addTextField(2,5,1);
```

```
  if (dlg.show()) {
    var result = new Object();
    result.serverName = serverName.text;
    result.replyTo = replyTo.text;
    result.interval = interval.text;
    return result;
  } else {
    return null;
  }
}
```

Dieser Dialog liefert ein Objekt mit den Properties *serverName*, *replyTo* und *interval* zurück. Diese Werte werden dann eingetragen.

```
  var params = queryParams();
  if (!params) {
    return;
  }

  var template = '<ruleset>\n  <base>\n    <name>NotifyWf</name>\n
<search>\n' +
    '      <name>"DIRECT"</name>\n      <value>"1"</value>\n
<mask>0</mask>\n' +
    '      <max>200</max>\n    </search>\n
<interval>INTERVAL</interval>\n  </base>\n' +
    '<rule>\n  <name>SendReminder</name>\n  <condition></condition>\n' +
    '  <script>\n    mail.setSmtpHost("SERVERNAME");\n' +
    '    notify.processAllUsers("REPLYTO", "Workflowbenachrichtigung",
true, true, true);\n' +
    '</script>\n</rule>\n<rule>\n  <name>Global Error Rule</name>\n
<condition>OnError</condition>\n' +
    '  <script></script>\n</rule>\n</ruleset>';

  template = template.replace( /SERVERNAME/, params.serverName );
  template = template.replace( /REPLYTO/, params.replyTo );
  template = template.replace( /INTERVAL/, params.interval );
```

Dieser Template-String wird in eine Datei geschrieben und anschließend in das Archiv übertragen.

```
  var payloadDir = workspace.directories.payloadDir;
  var file = utils.getUniqueFile(payloadDir, "NotifyWf.txt");
  FileUtils.writeStringToFile(file, template, "UTF-8");

  var sord = rules.prepareDocument(0);
  sord.name = "NotifyWf";
  rules.addDocument(sord, file);
```

Es gibt noch weitere Templates, die als Dateien in einem Exportdatensatz zur Verfügung stehen. Sie werden im ELOas Base-Verzeichnis in das Unterverzeichnis *Misc* übertragen. Da es sich bei diesem Unterverzeichnis nicht um ein Standardverzeichnis handelt, wird vor dem Import geprüft, ob es vorhanden ist und es wird dann bei Bedarf automatisch angelegt.

```
  try {
    var eloasMisc = archive.getElementByArcpathRelative(eloasBase.id,
"¶Misc");
  } catch(e) {
    // Misc Verzeichnis existiert noch nicht, jetzt anlegen
    var misc = eloasBase.prepareStructure("1");
    misc.name = "Misc";
    eloasMisc =eloasBase.addStructure(misc);
  }
```

In dieses Verzeichnis wird dann der Import der Templates durchgeführt.

```
  workspace.doImport(payloadDir + "\\ELOasMisc.zip", eloasMisc.id,
ImportOptionsC.GUIDS_KEEP, 0);
```

Zusätzlich müssen aber auch Skriptdateien für den ELO Java Client installiert werden. Diese liegen als weiterer Importdatensatz vor. Die Ermittlung des Zielverzeichnisses erfolgt wieder über die GUID.

```
var jcScriptFolder = archive.getElementByGuid('(E10E1000-E100-E100-E100-
E10E10E10E11)');

var payloadDir = workspace.directories.payloadDir.path;

workspace.doImport(payloadDir + "\\NotifyJc.zip", jcScriptFolder.id,
ImportOptionsC.GUIDS_KEEP, 0);
```

Das Verzeichnis *install.data* enthält die beiden Importdatensätze.

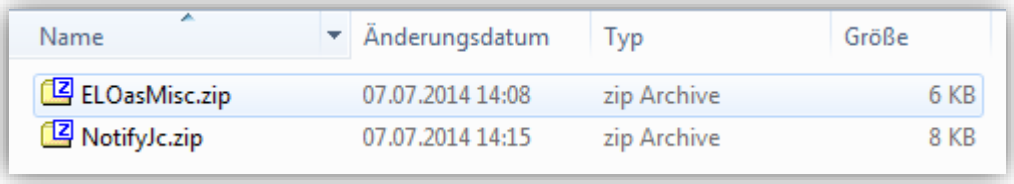


Abb. 6: Verzeichnis 'install.data'

Das komplette Skript sieht so aus:

```
doInstall();

function doInstall() {
  try {
    var jcScriptFolder = archive.getElementByGuid('(E10E1000-E100-E100-
E100-E10E10E10E11)');
  } catch(e) {
    workspace.showAlertBox("ELO Install", "Cannot find Java Client
Scripting Folder, Import aborted");
    return;
  }

  var payloadDir = workspace.directories.payloadDir.path;

  workspace.doImport(payloadDir + "\\NotifyJc.zip", jcScriptFolder.id,
ImportOptionsC.GUIDS_KEEP, 0);

  try {
    var eloasBase = archive.getElementByArcpath("¶Administration¶ELOas
Base");
  } catch(e) {
    workspace.showAlertBox("ELO Install", "Cannot find ELOas Base Folder,
Import aborted");
  }

  try {
    var eloasMisc = archive.getElementByArcpathRelative(eloasBase.id,
"¶Misc");
  } catch(e) {
    // Misc Verzeichnis existiert noch nicht, jetzt anlegen
    var misc = eloasBase.prepareStructure("1");
    misc.name = "Misc";
    eloasMisc =eloasBase.addStructure(misc);
  }

  workspace.doImport(payloadDir + "\\ELOasMisc.zip", eloasMisc.id,
ImportOptionsC.GUIDS_KEEP, 0);

  createRule(eloasBase);
}

function queryParams() {
  var dlg = workspace.createGridDialog("Notify - Installation", 3, 6);
  var panel = dlg.gridPanel;

  var labelText = '<html><div style="width:400px; padding:5px; background-
color: #f0f0f0">' +
        'Einstellungen für den EMail Versand' +
```

```
        '</div></html>';

  panel.addLabel(1,2,2, labelText);

  panel.addLabel(1,3,1, "Name des Mail-Servers");
  var serverName = panel.addTextField(2,3,1);

  panel.addLabel(1,4,1, "Reply-To Mail-Adresse");
  var replyTo = panel.addTextField(2,4,1);

  panel.addLabel(1,5,1, "Intervall - z.B. 06:00 für jeden Tag 6 Uhr
morgens");
  var interval = panel.addTextField(2,5,1);

  if (dlg.show()) {
    var result = new Object();
    result.serverName = serverName.text;
    result.replyTo = replyTo.text;
    result.interval = interval.text;
    return result;
  } else {
    return null;
  }
}

function createRule(base) {
  var rules = archive.getElementByArcpathRelative(base.id, "¶Rules");

  try {
    archive.getElementByArcpathRelative(rules.id, "¶NotifyWf");
    log.debug("Rule NotifyWf available, nothing to do.");
    return true;
  } catch(e) {
    log.debug("Rule NotifyWf not available, create now.");
  }

  var params = queryParams();
  if (!params) {
    return;
  }

  var template = '<ruleset>\n  <base>\n    <name>NotifyWf</name>\n
<search>\n' +
    '      <name>"DIRECT"</name>\n      <value>"1"</value>\n
<mask>0</mask>\n' +
    '      <max>200</max>\n    </search>\n
<interval>INTERVAL</interval>\n  </base>\n' +
    '<rule>\n  <name>SendReminder</name>\n  <condition></condition>\n' +
    '  <script>\n    mail.setSmtpHost("SERVERNAME");\n' +
    '    notify.processAllUsers("REPLYTO", "Workflowbenachrichtigung",
true, true, true);\n' +
```

```
    '</script>\n</rule>\n<rule>\n  <name>Global Error Rule</name>\n
<condition>OnError</condition>\n' +
    '  <script></script>\n</rule>\n</ruleset>';

  template = template.replace( /SERVERNAME/, params.serverName );
  template = template.replace( /REPLYTO/, params.replyTo );
  template = template.replace( /INTERVAL/, params.interval );

  var payloadDir = workspace.directories.payloadDir;
  var file = utils.getUniqueFile(payloadDir, "NotifyWf.txt");
  FileUtils.writeStringToFile(file, template, "UTF-8");

  var sord = rules.prepareDocument(0);
  sord.name = "NotifyWf";
  rules.addDocument(sord, file);
}
```

4.1 Verwendung von Callback-Aufrufen

Ab der Version 9.3 können auch Callback-Aufrufe in den eloinst-Skripten verwendet werden. Das ist in älteren Versionen nicht möglich, da der Aufruf nur in den fest installierten Skripten gesucht und aktiviert wurde und nicht in dem temporären eloinst-Skript.

```
panel.addButton(1,1,1, "MyButton", "onMyButtonPressed");
```

Ab der Version 9.3 wird die Funktion *onMyButtonPressed* im eloinst-Skript gesucht und aufgerufen.

4.2 Einfügen von Library-Dateien

Genauso wie in fest installierten Skripten können seit der Version 9.3 auch in eloinst-Skripten Includes eingefügt werden. Die Syntax für den Aufruf sieht so aus:

```
//@include lib_MyLibrary
```

Die eingebundenen Dateien müssen im ZIP-Paket im gleichen Verzeichnis wie die Skriptdatei liegen. Wenn es zu einer Library mehrere Includes gibt, wird sie trotzdem nur einmal eingebunden – an der Stelle des ersten Includes. Alle folgenden Includes werden ignoriert.

Beachten Sie bitte, dass in JavaScript die Reihenfolge der Funktionsdefinitionen und Aufrufe wichtig sein kann. Sie können die Includes also im Allgemeinen nicht an einer beliebigen Stelle einfügen. Welches die „richtige“ Stelle ist, müssen Sie selber entscheiden.

4.3 Einfügen von Konfigurationsdateien

Bei umfangreichen Skripten ist es oft sinnvoll, die Konfiguration von den eigentlichen Skripten zu trennen. Die Konfiguration wird im Projekt oft angepasst, die Skripte hingegen sollten austauschbar bleiben, damit Updates leichter installiert werden können.

Ab der Version 9.3 gibt es dafür das Kommando *@config*. Analog zu den Includes kann man eine externe Konfigurationsdatei hinterlegen, welche dann zur Laufzeit in das Skript geladen wird.

```
//@config myConfigVar myConfigFile.json
```

Die Konfigurationsdatei *myConfigFile.json* muss ebenfalls im gleichen ZIP-Verzeichnis wie die Skriptdatei liegen. Die Konfigurationsdaten müssen im JSON-Format vorliegen und werden in die Skript-Variable *myConfigVar* eingelesen.

```
{
  "prop1" : 4711,
  "prop2" : "Ein Test"
}
```

```
// JavaScript Funktionen
//@config myConfigVar myConfigFile.json
…
var objId = myConfigVar.prop1;
```

5 Installation von einer Webseite

Die automatische Installation aus einer Webseite heraus erfolgt über den elodms-Protokoll-Handler. Dieser wird durch das Programm *ELOactivateJc.exe* zur Verfügung gestellt und kann deshalb nur in einer Vollinstallation verwendet werden. Ein Webstart-Client kann nur mit der Drag-and-drop Variante arbeiten.

Ein Link innerhalb einer Webseite zur Installation besteht aus der Einleitung `elodms://im/`. Danach folgt die vollständige URL auf das Installationspaket. Die URL kann aber innerhalb einer anderen URL nicht im Klartext niedergeschrieben werden, deshalb wird sie Base64 kodiert.

```
<a href="elodms://im/aHR0cDovL3NydnRkZXYwMzo2MDEwL25vdGlmeS56aXA=">Modul
jetzt installieren</a>
```

Wenn ein Benutzer diesen Link anklickt, ruft der Browser den elodms-Protokoll-Handler (ELOactivateJc.exe) auf und übergibt die komplette URL. Dieser übergibt die Daten wiederum über die COM-Schnittstelle an den ELO Java Client. Dort wird aus der Base64-Kodierung die URL ermittelt, das Installationspaket heruntergeladen und entpackt. Anschließend wird das Installationsskript gestartet.

Beispiel für eine Installations-Webseite:

```
<html>
<head>
  <style>
    body {
      font-family:sans-serif,Tahoma,Arial;
      width: 900px;
    }

    h1 {
      background-color: #f0f0f0;
      padding: 5px;
    }

    .item {
      background-color: #f0f0f0;
      margin-bottom: 20px;
      padding: 5px;
    }
  </style>
</head>
<body>
  <h1>ELO Module - Installationsübersicht</h1>
  <p>Im Folgenden werden die verfügbaren ELO Zusatzmodule aufgeführt. Zur
Installation müssen Sie als
     Administrator im Java Client angemeldet sein. Die Module werden dann
automatisch installiert.</p>
```

```
  <p><small>Beachten Sie bitte, dass zur Installation der elodms
Protokollhandler aktiviert sein muss.</small></p>
  <h3>Java Client Module</h3>
  <p>Die folgenden Module stellen zusätzliche Funktionserweiterungen
innerhalb des Java Clients zur Verfügung.
     Es sind keine weiteren ELO Komponenten betroffen</p>

  <div class="item">
  <h4>GotoId - Gehe zu einer Archivposition</h4>
  <p>Dieses Skript zeigt einen Dialog zur Eingabe einer logischen Objekt-
Id, einer Objekt-GUID oder einer Datei-Id an.
     Nachdem der Benutzer den Dialog bestätigt hat, springt der Client an
die angegebene Stelle im Archiv.</p>
  <p><a
href="elodms://im/aHR0cDovL3NydnRkZXYwMzo2MDEwL2dvdG9pZC56aXA=">Modul jetzt
installieren</a></p>
  </div>

  <div class="item">
  <h4>Standardregister - Erweiterte Standardregisterfunktion</h4>
  <p>Dieses Skript erweitert die Standardregisterfunktion so, dass auch
komplette Strukturen mit
  Verschlagwortung und Unterstrukturen angelegt werden können.</p>
  <p><a href="elodms://im/aHR0cDovL3NydnRkZXYwMzo2MDEwL3Rlc3Quemlw">Modul
jetzt installieren</a></p>
  </div>

  <h3>ELO Automation Services Module</h3>
  <p>Die folgenden Module stellen zusätzliche Funktionen für den ELOas zur
Verfügung. Im Allgemeinen sind
     auch weitere Module von den Erweiterungen betroffen.</p>

  <div class="item">
  <h4>Notify - Benachrichtigung über aktive Workflowaufgaben versenden</h4>
  <p>Dieses Modul überwacht die Workflowtermine der Benutzer und versendet
bei Bedarf eine EMail mit
     einer Liste der aktiven Aufgaben. Jeder Benutzer kann individuell
konfigurieren, wie er benachrichtigt
     wird. Diese Konfiguration wird durch ein Java Client Skript verwaltet,
welches automatisch mit
     installiert wird.</p>
  <p><a
href="elodms://im/aHR0cDovL3NydnRkZXYwMzo2MDEwL25vdGlmeS56aXA=">Modul jetzt
installieren</a></p>
  </div>

</body>
</html>
```

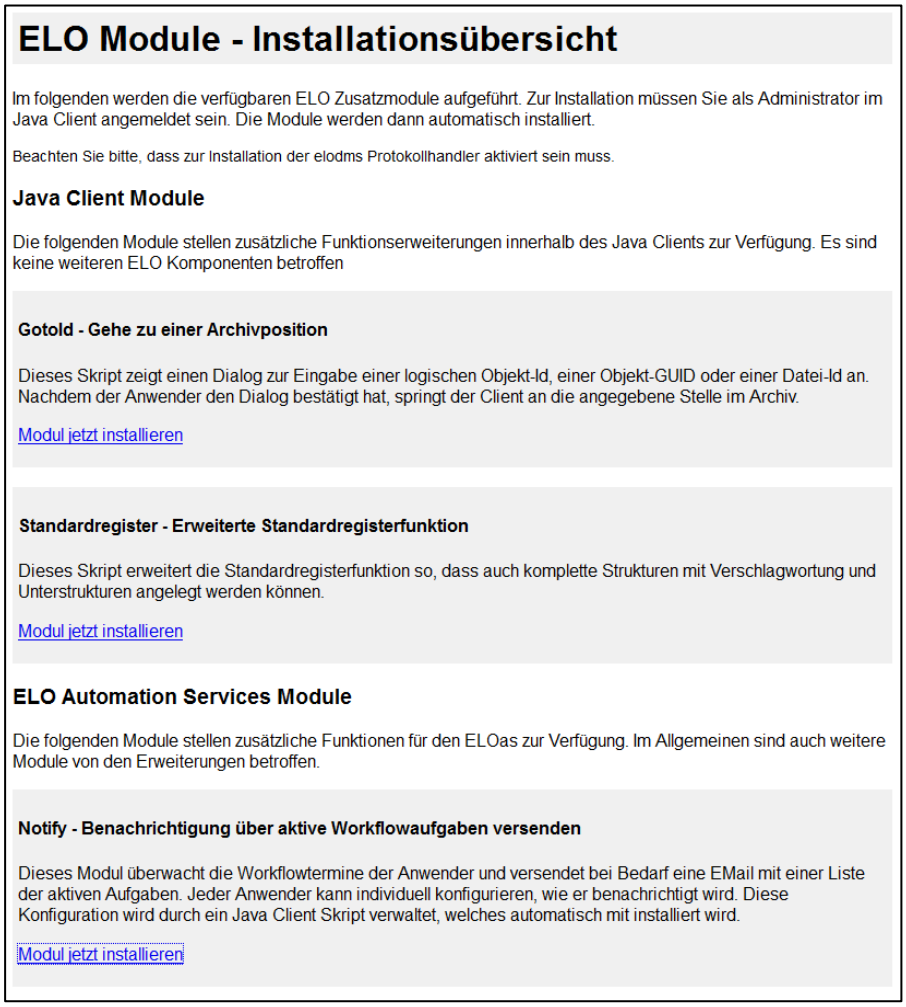


Abb. 7: Beispiel für eine Installations-Webseite

5.1 Ermittlung der Base64-URL

Die Umwandlung einer URL in das Base64-Format kann man einfach über ein kleines ELO Java Client Skript durchführen.

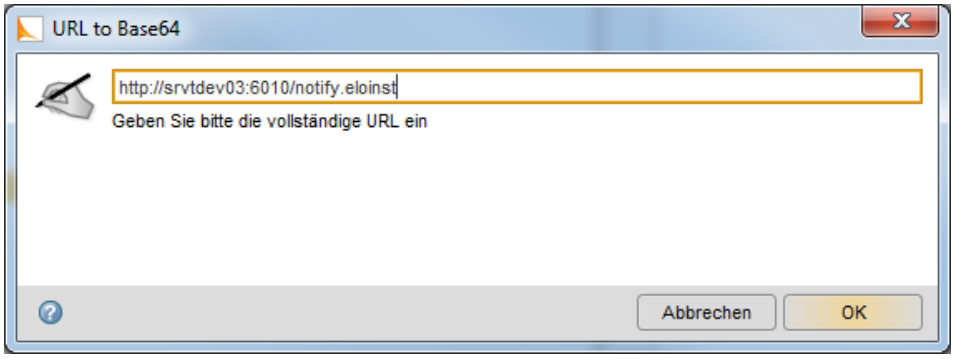


Abb. 8: Eingabe der URL

Dieses Skript meldet im Tab *Archiv* die Schaltfläche *URLtoB64* an. Wenn man diese aktiviert, wird über ein Eingabefenster die URL abgefragt.

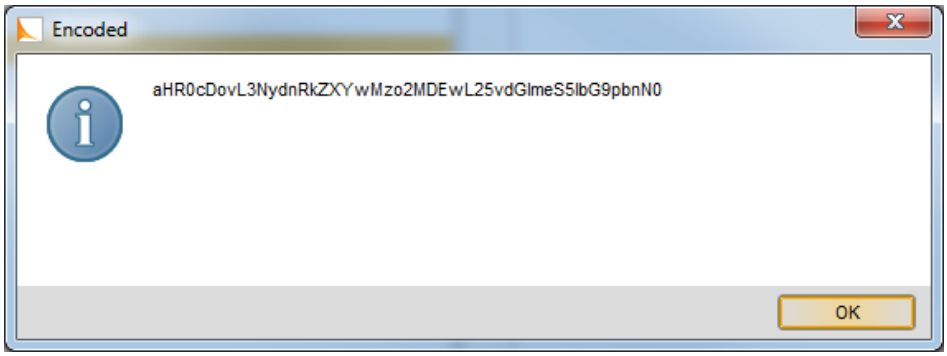


Abb. 9: URL in Base64-Darstellung

Beim Klick auf *Ok* wird aus der URL die Base64-Darstellung ermittelt und angezeigt. Weiterhin wird sie automatisch in die Windows-Zwischenablage übertragen, man kann sie also leicht mittels STRG-V in die Webseite eintragen.

```
function getScriptButton103Name()
{
 return "URLtoB64";
}

function getScriptButtonPositions()
{
  return "103,archive,information";
}

function eloScriptButton103Start() {
  var url = workspace.showInputBox("URL to Base64", "Geben Sie bitte die
vollständige URL ein", "", -1, -1, false, -1);
  if (url) {
    var base64 = new Packages.com.mindprod.base64.Base64();
    var url64 = base64.encode(url.getBytes());
    workspace.showInfoBox("Encoded", url64);

    var clipboard =
Packages.java.awt.Toolkit.getDefaultToolkit().getSystemClipboard();
    clipboard.setContents(new
Packages.java.awt.datatransfer.StringSelection(url64), null);
  }
}
```