Skriptprogrammierung
mit ELO*office*

Matthias Thiele
Leiter Entwicklung

ELO Digital Office GmbH

Skriptprogrammierung mit ELOoffice

Die Produktfamilie ELOoffice, ELOprofessional und ELOenterprise bietet schon seit vielen Jahren eine umfangreiche OLE Automation Schnittstelle zur projektbezogenen Erweiterung der Standardfunktionen. Die Dokumentation dieser Schnittstelle umfasst über 400 Seiten mit Beschreibungen der verfügbaren Funktionen und Parameter. Es gibt dort aber, neben ein paar Beispielen, keine Informationen zum prinzipiellen Aufbau von Skripten.

Dieses Buch soll anhand von einfachen und umfangreichen Beispielen aufzeigen, wie die OLE Automation Schnittstelle verwendet werden kann. Nebenbei werden auch ein paar allgemeine Hinweise gegeben, wie man robuste, leicht wartbare und wiederverwendbare Skriptbausteine erstellen kann.

Im Anhang befindet sich zusätzlich noch eine thematisch sortierte Liste der wichtigsten OLE Automation Funktionen. Sie erleichtert die Suche nach einer Funktion, wenn der Aufrufname nicht bekannt ist.

Inhalt

Vorwort

Dieses Buch wendet sich an Anwender, die bereits Grundkenntnisse in der Programmierung haben und nun in die Skriptentwicklung für ELOoffice einsteigen möchten. Es beginnt mit relativ einfachen Beispielen, die nach und nach zu umfangreichen Skripten erweitert werden. Viele Programmteile können als Anregung oder Vorlage für eigene Skripte genutzt werden. Vor allem die Skripte auf den hinteren Seiten können auch dem erfahrenen Skriptentwickler als Anregung oder Vorlage für eigene Entwicklungen dienen.

Skriptentwicklung soll einfach sein und dem Entwickler einen schnellen Zugang zur Lösung einfacher und mittelschwerer Probleme bieten. Das kann VB-Script gut leisten. Dabei besteht aber auch für den erfahrenen Entwickler das Risiko, dass er sich, nachdem er sich einmal einen ausreichenden Überblick verschafft hat, nicht weiter entwickelt. VB-Script bietet neben den einfachen Basisfunktionen durchaus auch anspruchsvollere Konstruktionen, die einem das Leben einfacher machen können. Aus diesem Grund sind in dem Buch auch ein paar Absätze zu allgemeinen Themen der Softwareentwicklung mit VB-Script eingestreut.

Die Beispiele sind für ELOoffice 9.0 erstellt worden. Auch wenn viele Skripte unter ELOoffice 8.0 lauffähig sind, gibt es doch ein paar Befehle, die in der älteren Version noch nicht zur Verfügung stehen. Aber auch ELOprofessional Anwender können viele Beispiele aus diesem Buch übertragen. Wenn Sie mindestens die Version 7.0 einsetzen, reicht im Normalfall der Austausch von „ELO.office“ in „ELO.professional“ für den Einsatz mit dem Windows Client aus.

Ein Teil der Skripte wurden im Anhang auch als Bibliotheksfunktionen aufbereitet und können direkt in eigene Programme eingebunden werden.

Zur Verwendung dieses Buchs

In den ersten Beispielen werden einfache Skripte vorgestellt, zusammen mit Erläuterungen zur Verwendung der Oberfläche und den notwendigen Einstellungen im ELOoffice. Deshalb sollten Sie diese auch dann durchlesen, wenn Sie mit der Skriptprogrammierung bereits vertraut sind. Die weiteren Beispiele sind unabhängig voneinander, sie können in beliebiger Reihenfolge bearbeitet werden.

Alle Skripte stehen auch auf unserer Supportseite zum Download zur Verfügung. Sie finden sie unter

http://www.eloit.de/info/demoscripts.zip

Gerade bei umfangreichen Skripten kann man sich damit die langwierige Tipparbeit ersparen. Dabei ist aber zu Beachten, dass gerade die umfangreicheren Skripte nicht „mal eben so“ ausgeführt und getestet werden können. Diese benötigen zum Teil spezielle Ablagemasken, die konfiguriert werden müssen oder spezielle Strukturen. Diese Anforderungen werden im Text zum Skripte jeweils beschrieben und müssen vor der Ausführung beachtet werden.

Programmbeispiele können Sie beim Durchblättern leicht an der Schriftart erkennen:

```
' Das ist ein Skript
Set ELO = CreateObject("ELO.office")
```

An verschiedenen Stellen werden auch wichtige Hinweise aufgeführt. Diese erkennen Sie ebenfalls schnell an der Formatierung:

> Hinweis: Wichtige Hinweise in diesem Buch sind durch diese Formatierung auch beim Durchblättern leicht zu erkennen.

Benötigte Systemumgebung

Die Beispiele benötigen zur Ausführung die VB-Script Version 5.0 oder neuer. Erstellt wurden sie unter der Version 5.8. Diese Version ist auf aktuellen Windows Versionen normalerweise vorhanden. Nur bei sehr alten Windows 2000 oder gar Windows 98 Rechnern wird das nicht der Fall sein. Auf diesen Rechnern kann ELOoffice 9.0 aber ohnehin nicht eingesetzt werden. Die Versionsnummer können Sie über ein kleines Skript nachprüfen:

```
MsgBox ScriptEngineMajorVersion & "." & ScriptEngineMinorVersion
```

Von der ELO Seite her wird die Programmversion ELOoffice 9.0 oder ELOprofessional 7.0 benötigt. Alle abgedruckten Beispiele sind für ELOoffice formuliert, die Anpassung für ELOprofessional beschränkt sich aber auf den unterschiedlichen CreateObject Aufruf.

Zu meiner Person

Zum Abschluss der Einleitung möchte ich noch ein paar Worte zu meiner Person los werden.

Ich habe an der RWTH Aachen Informatik studiert und anschließend freiberuflich verschiedene Firmen betreut. Diese Arbeit hat mir einen breiten Einblick in unterschiedliche Branchen, Arbeitsweisen und Kulturen geboten.

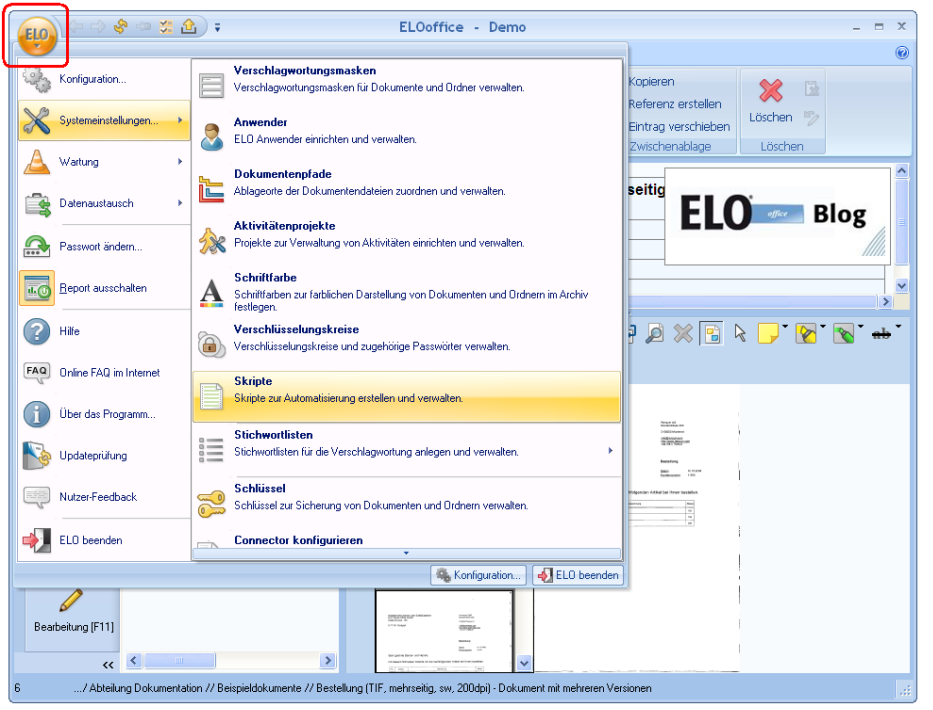
Über die Programmierung eines Hardware Treibers für eine Beschleunigerkarte im Imaging Bereich bin ich dann zum Dokumentenmanagement gekommen. Später habe ich als Angestellter zur Firma Leitz gewechselt, wo ich zuerst als Architekt und leitender Entwickler, später als Entwicklungsleiter, mit einem kleinen Team die erste ELOoffice Version erstellt habe. Diese ist aus strategischen Gründen als ELOoffice 2.0 auf den Markt gekommen, eine Version 1.0 hat es nie gegeben.

Nachdem die Firma Leitz von Esselte aufgekauft wurde, haben wir den Konzern im Rahmen eines Management Buy Out verlassen. Seit dieser Zeit

bin ich als technischer Geschäftsführer tätig. Trotz der zusätzlichen Management Tätigkeiten ist der Entwurf, Entwicklung und Support von Produkten immer noch ein wichtiger und interessanter Teil meiner Aufgaben.

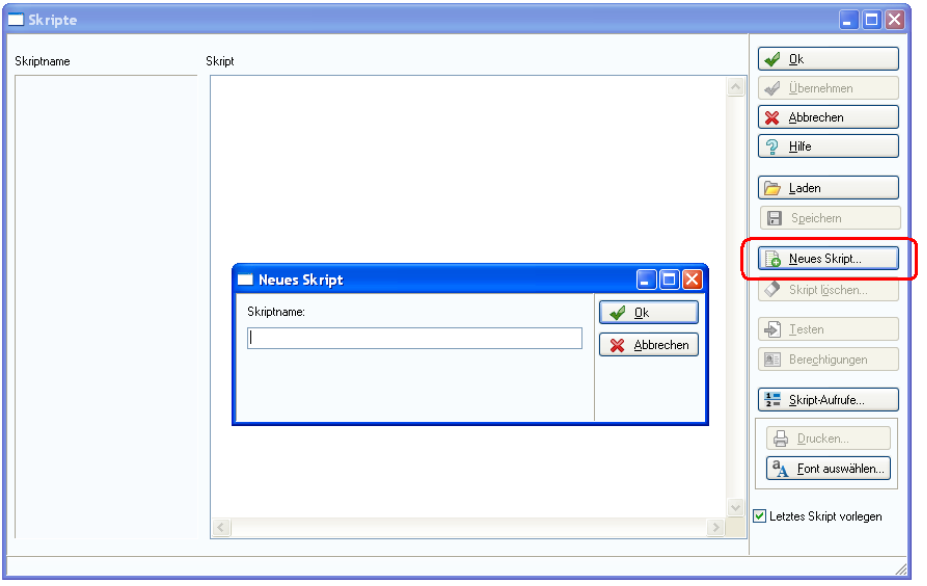
Erste Schritte

Fast jedes Programmierhandbuch beginnt mit einem „Hello World“ Beispiel, da möchte ich hier keine Ausnahme machen. Zur Eingabe von neuen Skripten öffnen Sie zuerst den ELO Skript Editor. Sie erreichen ihn über das Programmmenü unter „Systemeinstellungen“ – „Skripte“.



Ein erstes „Hello World“ Skript

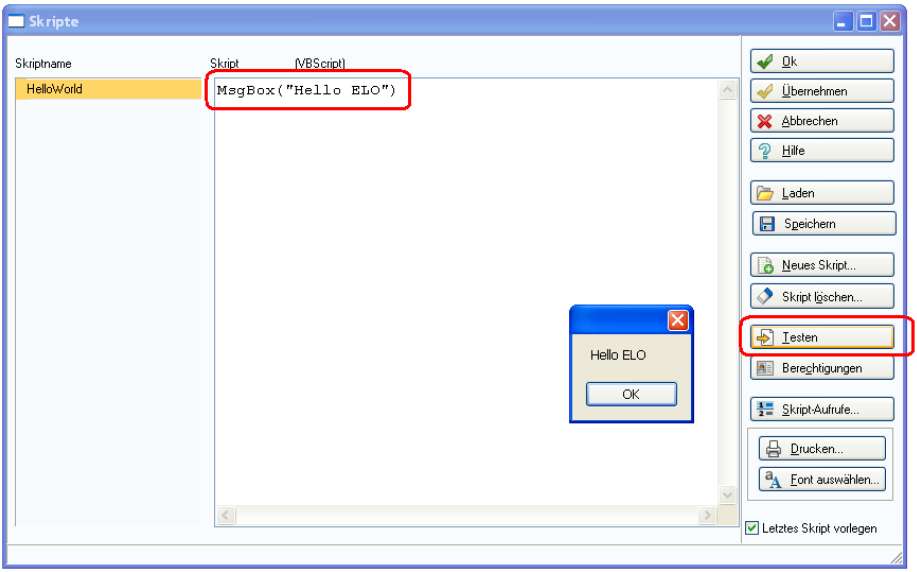
Klicken Sie in dem Dialog auf „Neues Skript…“ zur Anlage Ihres ersten Skripts. Geben Sie anschließend den Namen des neuen Skripts an: „HelloWorld“ und bestätigen Sie den Dialog mit Ok.



In dem Eingabefeld „Skript“ kann nun der Skriptcode für das erste Beispiel eingetragen werden. Geben Sie hier folgende Zeile ein:

MsgBox("Hello ELO")

Klicken Sie anschließend auf die Schaltfläche „Testen“. Hierüber wird das Skript sofort aufgerufen und es erscheint ein Dialog „Hello ELO“. Dieser kann durch einen Klick auf „Ok“ wieder geschlossen werden.



Damit haben Sie nun das erste Skript geschrieben und ausgeführt.

Hinweis: Da das Skript in einem eigenen Prozess läuft, sind die Meldungsausgaben unabhängig von der ELO Fensterreihenfolge. Wenn Sie bei einer bestehenden Ausgabe in das ELOoffice Fenster oder eine andere Anwendung klicken, dann „verschwindet“ das Meldungsfenster hinter den ELO Fenstern und ist nicht mehr einfach erreichbar. Wenn Sie ELOoffice über die Windows Tastenkombination <ALT>-Tab neu ansteuern, wird das Skriptfenster im Allgemeinen aber wieder in den Vordergrund geholt.

Skriptausführung unter ELOoffice

An dieser Stelle ist nun eine kurze Erläuterung notwendig, wie Skripte unter ELOoffice ausgeführt werden. Obwohl es den Anschein macht, als würde ELOoffice einen eigenen Skript Interpreter mitbringen, ist es in Wirklichkeit der Windows Scripting Host, der sich für die Aktionen verantwortlich zeigt. ELOoffice übergibt lediglich die Skriptdatei zur Ausführung und wartet das Ende der Operation ab. Aus diesem Grund ist es unwichtig, ob ein Skript aus ELOoffice heraus oder extern gestartet wurde. Die Schnittstellen und das Verhalten sind immer gleich.

Da der Windows Scripting Host keine Kenntnisse über die internen Daten aus ELOoffice besitzt, muss ein Skript also zuerst eine Verbindung aufbauen. ELO wird aus Skripten heraus genauso angesprochen wie jeder andere OLE Automation Server oder Active-X Control: es wird per CreateObject ein Zugriffsobjekt erstellt über das dann alle Operationen laufen. Dieses Objekt stellt alle Funktionen der ELO Automation Schnittstelle zur Verfügung, z.B. zur Abfrage und Bearbeitung der Verschlagwortung, Anzeige von Dokumenten oder auch zur Stammdatenverwaltung.

Ein verbessertes „Hello World“ Skript

Im letzten „HelloWorld“ Beispiel wurde eine wenig ansehnliche Systemmeldung über den Befehl MsgBox angezeigt. ELOoffice stellt über die OLE Schnittstelle eine bessere Variante bereit, welche sich in das Aussehen moderner Anwendungen besser einfügt. Damit diese verwendet werden kann, muss im neuen Beispiel ein ELOoffice Zugriffsobjekt erstellt werden. Dieses kann dann zur Anzeige der Meldung verwendet werden. Erzeugen Sie zuerst, wie oben beschrieben, ein neues Skript „HelloWorldELO“ mit folgendem Inhalt:

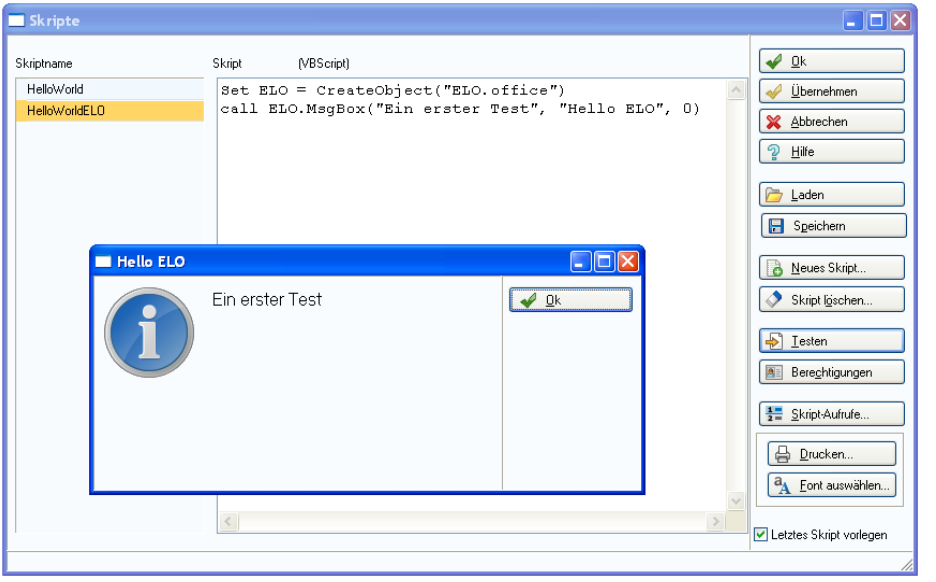
```
Set ELO = CreateObject("ELO.office")
call ELO.MsgBox("Ein erster Test", "Hello ELO", vbOkOnly)
```

In der ersten Zeile wird mittels CreateObject das Verbindungsobjekt zu ELOoffice erzeugt. Dieser Aufruf enthält einen Parameter, der die Applikation benennt, zu der eine Verbindung aufgenommen wird, im Beispiel „ELO.office“. Wir werden später aber auch noch andere Verbindungsobjekte zu anderen Applikationen erzeugen. Wie diese Benannt sind und welche Funktionen zur Verfügung gestellt werden, können Sie der Dokumentation der jeweiligen Anwendungen entnehmen.

In der zweiten Zeile wird dann anschließend eine „Message Box“ aufgerufen. Dieser Aufruf hat drei Parameter:

1. Den Nachrichtentext
2. Einen Fenstertitel
3. Zusätzliche Informationen

Die zusätzlichen Informationen werden in diesem einfachen Beispiel nicht verwendet, die Angabe vbOkOnly erzeugt einen einfachen Dialog, der mit Ok bestätigt werden kann.



Die Verwendung der ELO MessageBox statt der Standard VBS MessageBox hat auch den Vorteil, dass nun die Fensterreihenfolge besser geregelt ist. Dieser Dialog kann nun nicht mehr hinter dem ELO Fenster verschwinden und die weitere Programmausführung blockieren.

Weitere Optionen für die MsgBox

Diese Message Box kann auch unterschiedliche Schaltflächen anzeigen, z.B. Ok – Abbruch oder Ja – Nein. Diese Anzeige wird über den Flags Wert gesteuert, eine Liste der verfügbaren Werte finden Sie im Anhang A. Weiterhin können auf der linken Seite unterschiedliche Grafiken eingeblendet werden. Auch diese Werte finden Sie im Anhang.

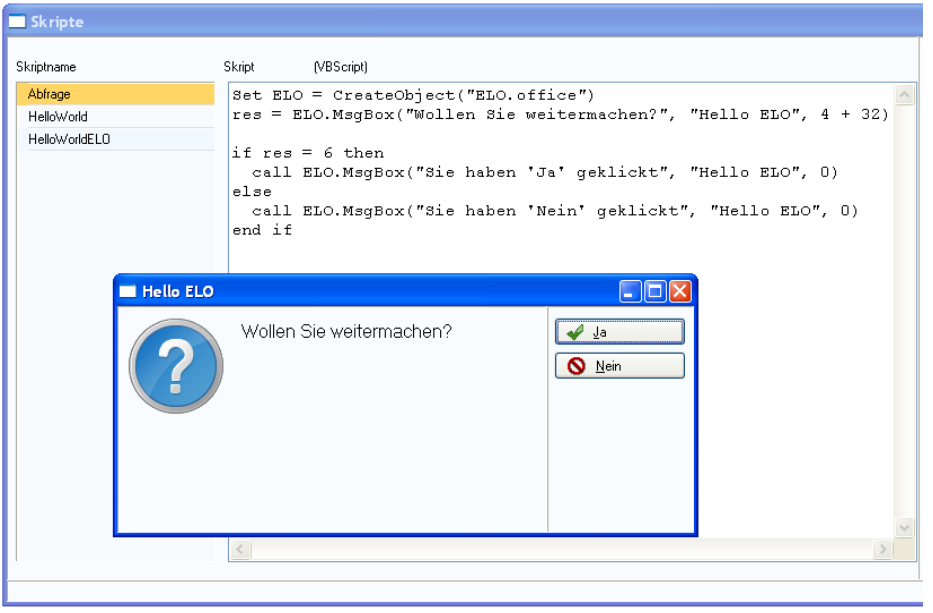
Das einfache Beispiel soll nun zu einer Abfrage ausgebaut werden. Hierzu muss die Message Box mit Schaltflächen für Ja – Nein versehen werden und der Rückgabewert geprüft werden. Legen Sie nun ein neues Skript „Abfrage“ an und tragen Sie folgende Programmzeilen ein:

```
Set ELO = CreateObject("ELO.office")
res = ELO.MsgBox("Wollen Sie weitermachen?", _
                 "Hello ELO", vbYesNo Or vbQuestion)

if res = vbYes then
  call ELO.MsgBox("Sie haben 'Ja' geklickt", "Hello ELO", vbOkOnly)
else
  call ELO.MsgBox("Sie haben 'Nein' geklickt", "Hello ELO", vbOkOnly)
end if
```

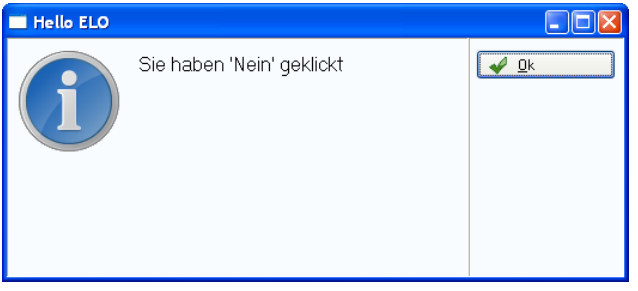
Im Flags Parameter der Message Box gibt es nun zwei Werte. Die Konstante vbYesNo sorgt für eine Anzeige der „Ja – Nein“ Schaltflächen und die Konstante vbQuestion blendet das Fragezeichen ein. Die beiden Werte können durch ein + oder ein Or miteinander verbunden werden. Im Anhang A finden Sie weitere Werte für die Anzeige und die Rückgabewerte.

Hinweis: lange Zeilen können Sie in VBS mithilfe eines Unterstrichs am Zeilenende umbrechen. Achten Sie aber darauf, dass der Unterstrich wirklich das letzte Zeichen vor dem Zeilenende ist. Danach können Sie in der nächsten Zeile weiter schreiben.

Die Anwenderauswahl wird in der Variablen res gespeichert und kann dann später im Programmablauf für Entscheidungen verwendet werden.



Nach einen Klick auf die Schaltfläche „Nein“:



„Gehe zu“ Skript zur Dokumentenanzeige

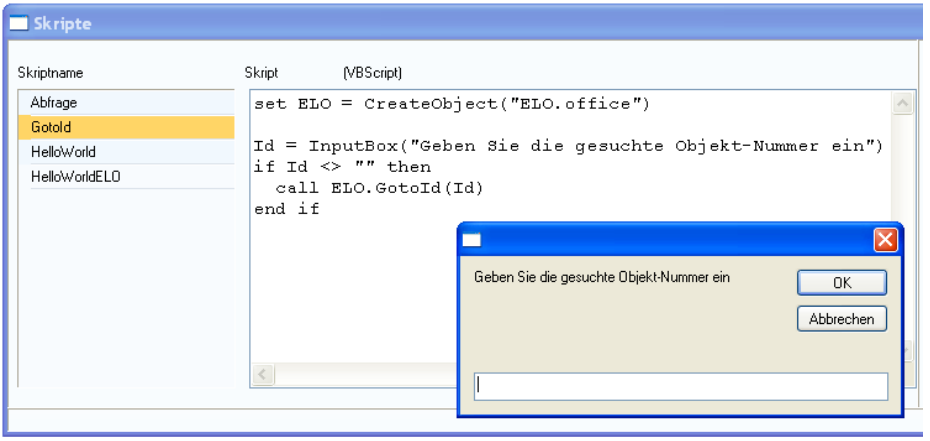
Das folgende Beispiel führt nun zum ersten Mal nützliche Aktionen im ELOoffice Archiv aus. An vielen Stellen in den ELO Reports wird auf die internen ELO Objektnummern verwiesen. Wenn man nun im Archiv nachsehen will, um welches Dokument es sich dabei handelt, ist das relativ aufwendig. Diese Suche kann man aber einfach über ein Skript abbilden. Hierfür gibt es den Befehl GotoId, der in der Archivansicht auf ein Ordner oder Dokument mit der angegebenen Objektnummer wechselt.

Einfache GeheZu Skriptversion

In den folgenden Zeilen finden Sie eine erste einfache Skriptversion um zu einer Objektnummer das zugehörende Dokument oder Ordner anzuzeigen.

```
set ELO = CreateObject("ELO.office")

Id = InputBox("Geben Sie die gesuchte Objektnummer ein")
if Id <> "" then
  call ELO.GotoId(Id)
end if
```

Bei dem Befehl InputBox handelt es sich um eine VB-Script Standardfunktion. Über sie kann eine Anwendereingabe abgefragt werden. Die Eingabe wird in der Variablen Id gespeichert. Der nachfolgende GotoId Befehl von ELO wird nur ausgeführt, wenn der Anwender eine Eingabe ausgeführt hat.



Dieses einfache Beispiel behandelt keine Fehleingaben. Es ist z.B. möglich, dass ein Anwender eine Nummer eingibt, die nicht im Archiv vorhanden ist oder aufgrund von Berechtigungseinschränkungen nicht erreichbar ist. In diesem Fall wird der Befehl nicht ausgeführt. Weiterhin ist es möglich, dass ein Anwender ungültige Zeichenfolgen, z.B. Buchstaben, eingibt. In einem „richtigen“ Programm sollten solche Fehlerzustände erkannt werden und dem Anwender aussagekräftige und hilfreiche Meldungen gegeben werden.

Version 2 mit zusätzlichen Fehlermeldungen

Eine etwas robustere Version des Skripts könnte so aussehen:

```
set ELO = CreateObject("ELO.office")

UserInput = InputBox("Geben Sie die gesuchte Objektnummer ein")
On Error Resume Next
Id = CLng(UserInput)
if Err.Number <> 0 then
  call ELO.MsgBox("Bitte geben Sie nur Ziffern ein.", "ELO", 0)
else
  if Id > 0 then
    res = ELO.GotoId(Id)
    if res < 0 then
      call ELO.MsgBox("Die Objektnummer wurde nicht gefunden",_
                      "ELO", 0)
```

```
    end if
  end if
end if
```

Hier wird nun geprüft, ob der Anwender eine gültige Nummer eingegeben hat und ob das Objekt im Archiv aufgefunden wurde. Zu den beiden möglichen Fehleingaben erhält der Anwender eine entsprechende Fehlermeldung.

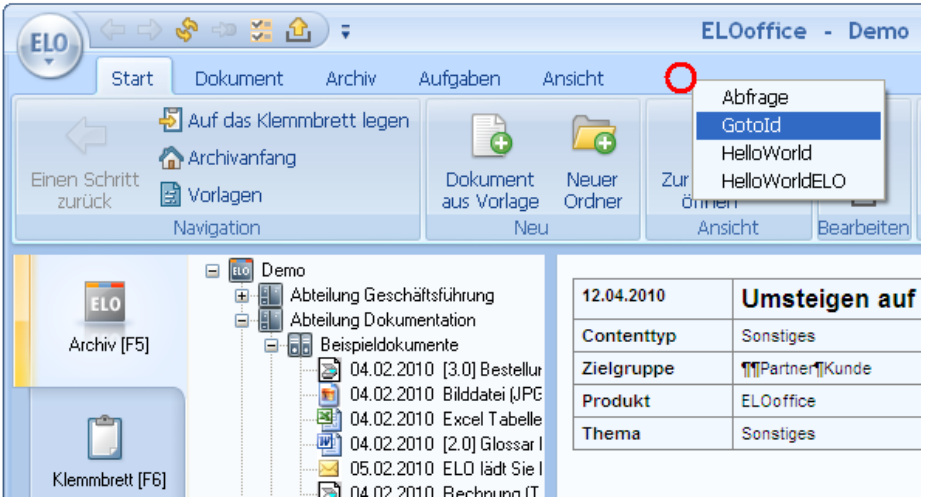
Anmerkung: wenn ein Zahlenwert eingegeben wird, der nicht als LONG Wert darstellbar ist, wird die etwas irreführende Fehlermeldung „Bitte geben Sie nur Ziffern ein.“ ausgegeben. Als Übung können Sie das Skript so erweitern, dass zu lange Zahlen von Texteingaben unterschieden werden.

Skriptaufruf aus der Multifunktionsleiste oder dem Kontextmenü

Nachdem das Skript getestet ist und wie gewünscht funktioniert, soll es den Anwendern zur Verfügung gestellt werden. Hierzu gibt es die Möglichkeit, dass eigene Skripte sowohl in dem Kontextmenü wie auch in den Schaltflächen der Multifunktionsleiste angeboten werden können.

Aufruf aus dem Kontextmenü

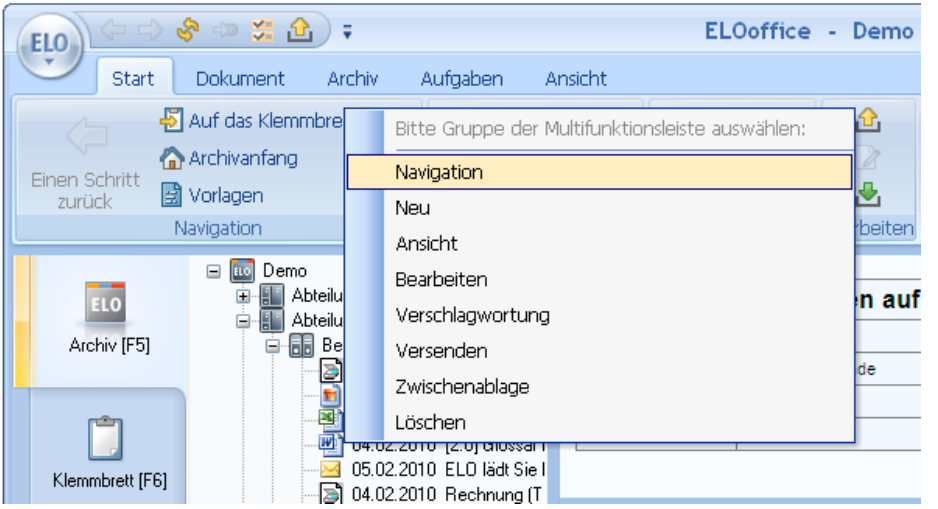
Zum einfachen Aufrufen eines Skripts können Sie mit der rechten Maustaste in die Tab-Leiste der Multifunktionsleiste klicken und aus der nun erscheinenden Liste aller Skripte das gewünschte auswählen. Es wird dann sofort ausgeführt.



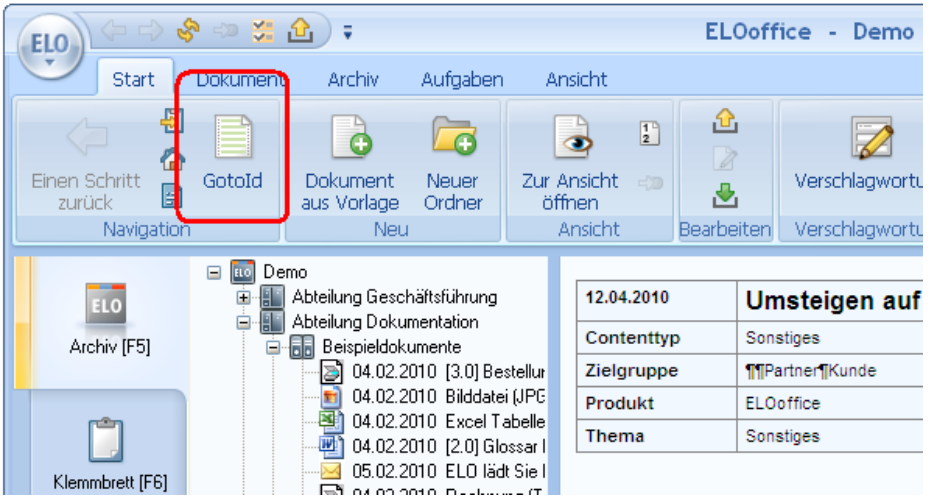
Diese Methode setzt aber voraus, dass man weiß, wo man klicken muss. Zudem arbeiten eine Menschen nur ungern mit Kontextmenüs. Aus diesem Grund gibt es noch eine andere Variante, die den Aufruf komfortabel über die Multifunktionsleiste zur Verfügung stellt.

Aufruf aus der Multifunktionsleiste

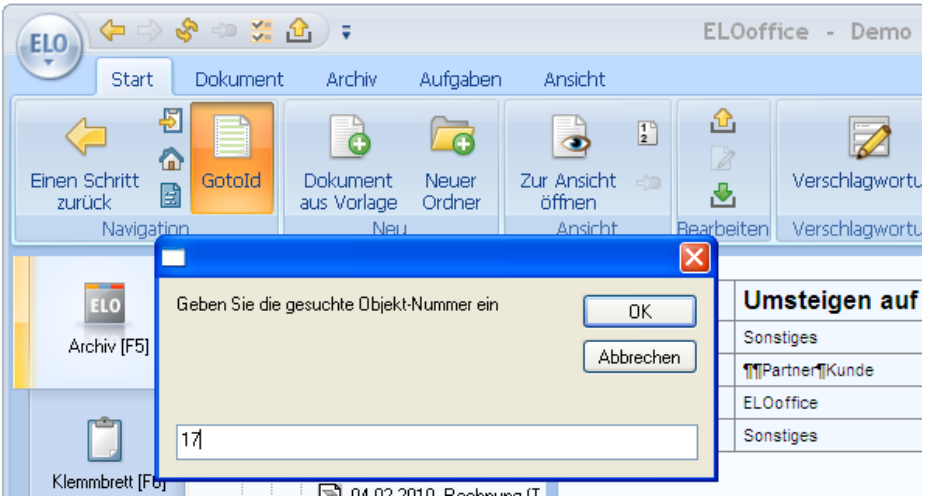
Zum Einfügen in die Multifunktionsleiste wählen Sie zuerst den gewünschten Zieltab aus (Start, Dokument, Archiv…) und klicken Sie ebenfalls mit der rechten Maustaste auf die Tab-Leiste. Es erscheint nun eine Liste aller verfügbaren Skripte. Wählen Sie nun das gewünschte Skript (im Beispiel GotoId) **mit gedrückter** <STRG>-Taste aus. Nun erscheint eine Auswahl aller verfügbaren Gruppen aus dem aktuellen Tab.



Wählen Sie auch hier den gewünschten Eintrag aus. Das Skript GotoId würde im Start-Tab am Besten in die Gruppe „Navigation“ passen. Nachdem Sie diesen Eintrag ausgewählt haben, finden Sie in der Multifunktionsleiste eine neue Schaltfläche „GotoId“ vor.

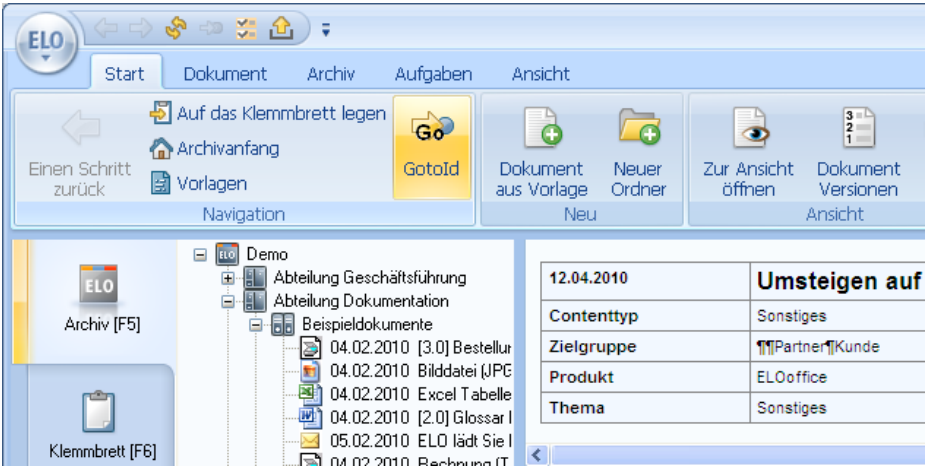


Der Anwender kann das Skript nun durch einfaches Anklicken aufrufen.



Skript-Schaltflächen haben als Voreinstellung alle das gleiche Icon mit dem grün-weißen Computerpapier. Sie können aber auch eigene Icons hinterlegen. Dazu zeichnen Sie das gewünschte Icon in einer Größe von 32 * 32 Punkten und speichern es als Bitmap Datei im ELO …\Postbox\EloScripts Verzeichnis mit gleichem Namen wie das Skript ab (also zu GotoId.vbs ein Bild GotoId.bmp).

Wenn Sie ELOoffice nun neu starten, wird statt des Standard Icons Ihr eigenes Skript Icon angezeigt.



Hinweis: Beachten Sie bitte, dass der Punkt in der linken unteren Ecke als Referenzpunkt für die transparente Farbe verwendet wird. Alle Bereiche mit diesem Farbton werden also durchsichtig gezeichnet, so dass an diesen Stellen die Multifunktionsleiste sichtbar wird.

Eine bessere InputBox

Die InputBox aus dem Beispiel hat das gleiche Problem wie die ursprüngliche MsgBox – sie wird durch einen wenig ansprechenden Systemdialog des Windows Scripting Host dargestellt. Aber auch hier gibt es eine Lösung im ELO Rahmen. Über die OLE Automation Schnittstelle wird eine Funktion zur Erstellung einfacher Dialoge angeboten. Diese besitzen dann das ELO Look And Feel und somit das Aussehen einer modernen Applikation. Da diese Dialoge Teil des ELO Programms sind, kann es auch nicht so einfach passieren, dass sie hinter dem ELO Hauptfenster verschwinden und die weitere Programmausführung blockieren.

So ein ELO Dialog wird mittels der Funktionsgruppe AutoDlg erstellt. Zuerst wird mit CreateAutoDlg ein leerer Dialog erzeugt, dort werden dann mit

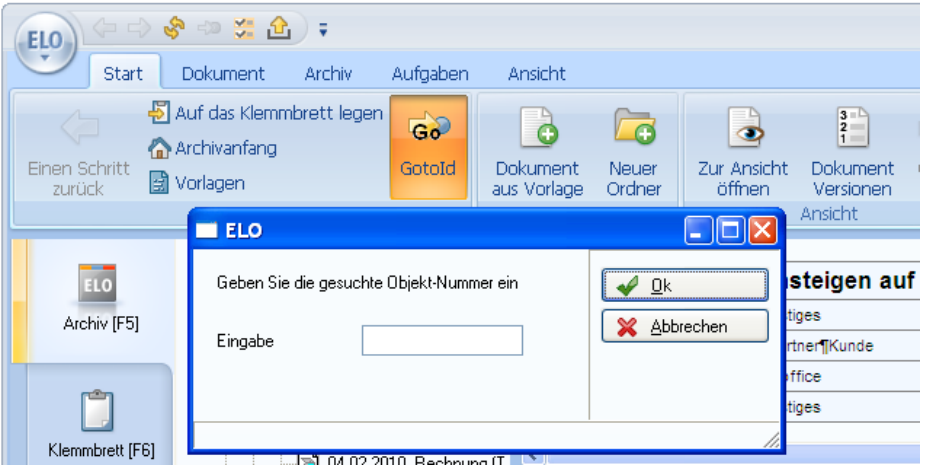
AddAutoDlgControl die Texte und Eingabezeilen erzeugt und anschließend wird der Dialog mit ShowAutoDlg angezeigt. Die Anwendereingabe kann nach dem Aufruf über GetAutoDlgValue abgefragt werden. Eine verbesserte InputBox mit ELO Mitteln könnte dann so aussehen:

```
function EloInputBox(Message, Titel)
  ELO.CreateAutoDlg(Titel)
  call ELO.AddAutoDlgControl(1, 0, Message, "")
  call ELO.AddAutoDlgControl(4, 2, "Eingabe", " ")
  if ELO.ShowAutoDlg = 1 then
    EloInputBox = ELO.GetAutoDlgValue(2)
  else
    EloInputBox = ""
  end if
end function
```

Das eigentliche GotoId Skript bleibt nahezu unverändert. Lediglich der Aufruf von InputBox wird durch die EloInputBox ersetzt und die oben aufgeführte Funktion EloInputBox hinzugefügt:

```
set ELO = CreateObject("ELO.office")

UserInput = EloInputBox("Geben Sie die gesuchte Nummer ein", "ELO")
if UserInput <> "" then
  On Error Resume Next
  Id = CLng(UserInput)
  if Err.Number <> 0 then
    call ELO.MsgBox("Bitte geben Sie nur Ziffern ein.", "ELO", 0)
  else
    if Id > 0 then
      res = ELO.GotoId(Id)
      if res < 0 then
        call ELO.MsgBox("Die Objekt-Id wurde nicht gefunden", "ELO", 0)
      end if
    end if
  end if
end if

function EloInputBox(Message, Titel)
  ELO.CreateAutoDlg(Titel)
  call ELO.AddAutoDlgControl(1, 0, Message, "")
  call ELO.AddAutoDlgControl(4, 2, "Eingabe", " ")
```

```
  if ELO.ShowAutoDlg = 1 then
    EloInputBox = ELO.GetAutoDlgValue(2)
  else
    EloInputBox = ""
  end if
end function
```



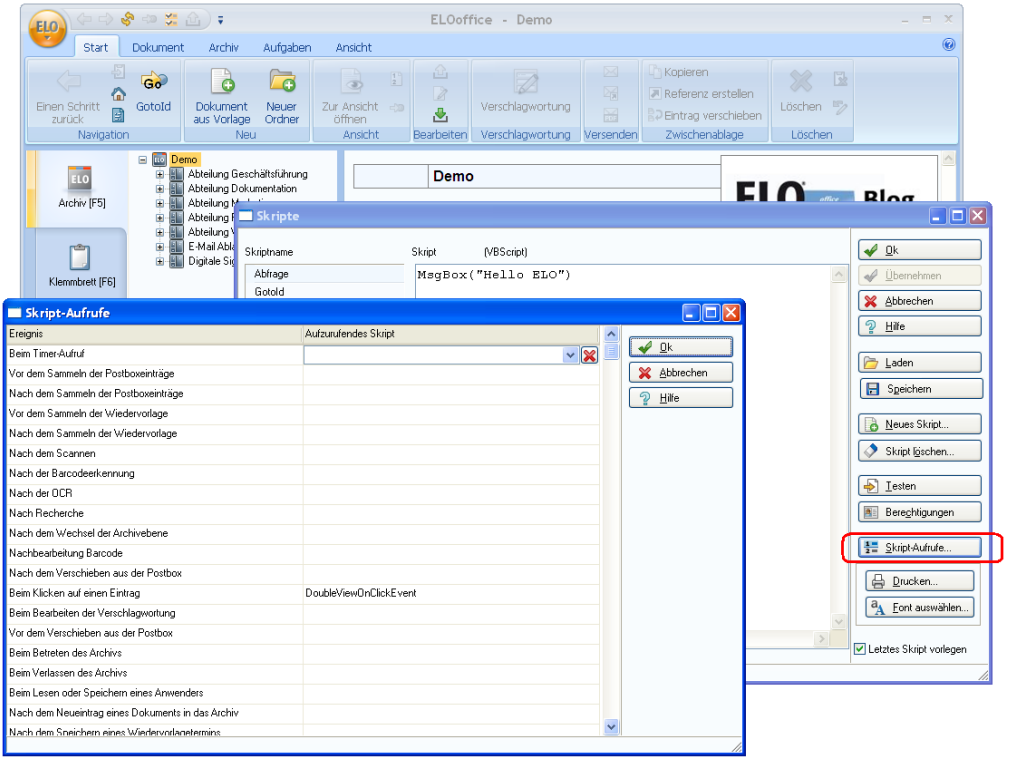
Diese finale Version hat nun ein zeitgemäßes User Interface und sinnvolle Eingabekontrollen und Fehlermeldungen. Dem Administrator kann dieses Skript bei der Auswertung von Reportdateien eine wertvolle Hilfe sein.

Ereignisgesteuerte Skriptausführung

Im letzten Abschnitt haben Sie gesehen, wie man Skripte erstellt und diese dem Anwender über eine Schaltfläche zur Verfügung stellen kann. Diese Vorgehensweise ist dann sinnvoll, wenn der Anwender selber entscheiden muss, ob und wann ein Skript verwendet wird. In manchen Situationen ist es aber wünschenswert, dass ein Skript immer dann automatisch ausgeführt wird, wenn ein bestimmtes Ereignis eintritt. So kann man z.B. vor jedem Speichervorgang bestimmte Plausibilitätskontrollen durchführen. Wenn man diese Prüfung auf eine Schaltfläche legt, dann ist das zum Einen für den Anwender aufwendig, da er ja vor jedem Speichervorgang extra noch mal dieses Skript anklicken muss. Zum Anderen ist es aber auch für den Prozess riskant, da es eine hohe Wahrscheinlichkeit gibt, dass die Prüfung hin und wieder mal vergessen wird. Für diese Situation gibt es die ereignisgesteuerten Skriptaufrufe.

Ereignisgesteuerte Skriptaufrufe gibt es beim Betreten oder Verlassen eines Archivs, beim Bearbeiten der Verschlagwortung, beim Anzeigen oder Verschieben eines Eintrags, beim Suchen und noch an vielen anderen Stellen mehr. Obwohl die Liste lang ist, ist es im Normalfall offensichtlich, welches Skriptereignis für ein spezielles Problem benötigt wird und verwendet werden kann.

Den Dialog zur Konfiguration dieser Skripte erreichen Sie aus der Skriptbearbeitung heraus über die Schaltfläche „Skript-Aufrufe“.



Hinweis: Bei ereignisgesteuerten Skripten sollten Sie bedenken, dass diese im Arbeitsablauf des Anwenders ausgelöst werden. Sie sollten dort also keine lang laufenden Aktionen ausführen, da Sie so den Arbeitsfluss erheblich stören würden.

Beim Timer-Aufruf	Dieses Ereignis wird regelmäßig im Hintergrund aufgerufen. Es ist aber nur aus Kompatibilitätsgründen zu älteren Versionen vorhanden und sollte nicht für Neuentwicklungen verwendet werden.

Vor dem Sammeln der Postboxeinträge	Wenn der Anwender die Postbox aktualisiert, werden die Dateien aus der Serverpostbox, dem Tiff- und PDF Druckerpfad und dem Netzwerkscannerpfad gesammelt. Dieses Ereignis wird vor dem Sammeln aufgerufen. Hier kann man z.B. ungenutzte Dateien entfernen, so dass sie nicht mehr in der Postboxliste aufgeführt werden.
Nach dem Sammeln der Postboxeinträge	Nachdem die Dateiliste in der Postbox aktualisiert wurde, wird dieses Ereignis zur Nachbearbeitung der Postboxliste aufgerufen.
Vor dem Sammeln der Wiedervorlage	Der Anwender oder eine Programmaktion hat eine Aktualisierung der Aufgabenansicht ausgelöst.
Nach dem Sammeln der Wiedervorlage	Die Terminliste der Aufgabenansicht wurde neu eingelesen.
Nach dem Scannen	Nach jeder eingelesenen Seite vom Scanner wird dieses Ereignis aufgerufen. Auch bei geklammerten Dokumenten wird es für jede einzelne Seite aktiviert.
Nach der Barcodeerkennung	Falls Barcodeinformationen nachbearbeitet werden müssen (z.B. weitere Indexfelder aus einer Datenbank übernehmen), kann man dieses Ereignis zum Skriptaufruf verwenden. Es wird einmal am Ende der kompletten Erkennung aufgerufen.
Nach der OCR	Statt der Barcodeerkennung kann auch eine OCR Verarbeitung in der Maskendefinition hinterlegt werden. In diesem Fall wird dieses Ereignis statt „Nach der Barcodeerkennung“ aufgerufen.

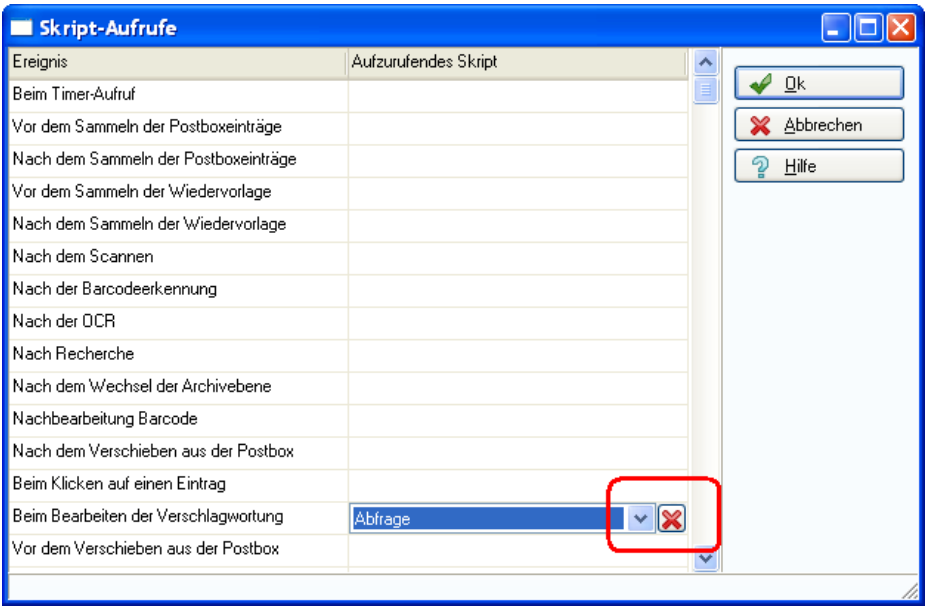
Nach Recherche	Nachdem eine Suche durchgeführt wurde und die Trefferliste gefüllt ist, wird dieses Skript zur Nachbearbeitung dieser Liste aktiviert.
Nach dem Wechsel der Archivebene	Dieses Ereignis ist nur aus Kompatibilitätsgründen zu älteren Versionen vorhanden und sollte nicht für Neuentwicklungen verwendet werden.
Nachbearbeitung Barcode	Falls Barcodeinformationen nachbearbeitet werden müssen (z.B. weitere Indexfelder aus einer Datenbank übernehmen), kann man dieses Ereignis zum Skriptaufruf verwenden. Es wird nach jeder einzelnen Barcodeseite aufgerufen.
Nach dem Verschieben aus der Postbox	Wenn ein Dokument aus der Postbox in das Archiv verschoben wurde, wird dieses Ereignis zur Nachbearbeitung der Ablage aufgerufen.
Beim Klicken auf einen Eintrag	Wenn der Anwender im Archiv, Treffer oder Aufgabenliste einen neuen Eintrag auswählt, wird dieses Ereignis aufgerufen. Hier kann z.B. Einfluss auf das anzuzeigende Dokument genommen werden.
Beim Bearbeiten der Verschlagwortung	Das ist ein Sammelereignis welches zu verschiedenen Zeitpunkten der Verschlagwortungsbearbeitung aufgerufen wird (z.B. zum Start oder Ende, Betreten oder Verlassen von Eingabefeldern). Die unterschiedlichen Zustände werden über das Property ActionKey erkannt.
Vor dem Verschieben aus der Postbox	Bevor ein Dokument aus der Postbox verschoben wird, kann über dieses Ereignis die Dokumentendatei noch beeinflusst werden.

Beim Betreten des Archivs	Wenn der Anwender nach der Anmeldung das Archiv betritt, wird dieses Ereignis aufgerufen. Hier können Initialisierungen für andere Skriptaufrufe stattfinden oder die Startansicht beeinflusst werden.
Beim Verlassen des Archivs	Vor dem Beenden des ELO Clients kann ein Skript hier Systeminformationen sichern oder aufräumen.
Beim Lesen oder Speichern eines Anwenders	Wenn die Anwenderverwaltung überwacht werden muss, kann das über dieses Skriptereignis erfolgen. Hier kann man z.B. für neue Anwender automatisch einen eigenen „Home“ Ordner anlegen lassen.
Nach dem Neueintrag eines Dokuments in das Archiv	Dokumente können auf unterschiedliche Wege in das Archiv gelangen. Wenn generelle Konsistenzüberprüfungen notwendig sind, kann man hier ein Skript einfügen.
Nach dem Speichern eines Wiedervorlagetermins	Nachdem ein Anwender einen Wiedervorlagetermin gespeichert hat, wird dieses Skriptereignis aufgerufen. Hier können z.B. Synchronisationen mit externen Kalendern erfolgen.
Vor dem Löschen eines Wiedervorlagetermins	Wenn ein Termin gelöscht wird, kann ein Skript diesen Termin auch aus anderen Kalendern löschen. Der Aufruf erfolgt vor dem Löschen damit die Informationen des Termins noch zur Verfügung stehen.
Vor dem Exportieren / Importieren eines Eintrags	Wenn Dokumente beim Export oder Import automatisch überprüft werden sollen, kann das hier erfolgen.

Vor der Anzeige eines Dokuments	Bevor eine Datei zur Anzeige an das interne Preview übergeben wird, kann diese hier bei Bedarf geprüft, verändert oder ergänzt werden.
Beim Aus-/Einchecken eines Dokuments	Beim Auschecken wird die Dokumentendatei aus dem Archiv in den CheckOut Ordner kopiert. Beim CheckIn von dort aus in das Archiv zurück übertragen. Dieses Ereignis bietet die Möglichkeit zu beiden Zeitpunkten auf die Datei Einfluss zu nehmen.
Beim Bearbeiten einer Haftnotiz	Wenn eine neue Haftnotiz angelegt wird oder eine bestehende Haftnotiz verändert wird, kann sie über dieses Ereignis nachbearbeitet werden.
Beim Eintragen/ Verschieben einer Objektreferenz	Wenn ein Ordner oder Dokument neu in einem Ordner abgelegt wird oder von einem anderen Ordner verschoben wird, kann hier eine Nachbearbeitung erfolgen, z.B. eine Anpassung der Verschlagwortung oder Berechtigung.
Vor dem Sammeln der Nachfolgerliste	Bevor zu einem Ordner die Liste der Untereinträge gesammelt wird, können von hier aus vorbereitende Aktionen durchgeführt werden.
Beim Lesen/ Schreiben einer Aktivität	Wenn ein Aktivitätentermin gelesen oder geschrieben wird, kann ein Skript den Inhalt ergänzen oder abändern.
Beim Viewer Export	Beim Export eines Archivs für den freitragenden Viewer können über dieses Skriptereignis weitere Dateien in den Zielbereich übertragen werden.

Vor dem Sammeln der Rechercheliste	An dieser Stelle kann ein Skript den automatisch generierten SQL Code an bestimmte Gegebenheiten anpassen um Optimierungen oder weitere Sucheinschränkungen zu erreichen.
Vor der HTML Verschlagwortungsanzeige	Wenn die direkte Verschlagwortungsanzeige über dem Dokument aktiviert ist, kann das Skript an dieser Stelle die Anzeigeinformation abändern.
Beim Löschen eines Eintrags	Wenn ein Ordner oder Dokument gelöscht wird, kann ein Skript zusätzliche Aufräumarbeiten veranlassen.

Wenn Sie ein Skriptereignis einrichten wollen, müssen Sie zuerst das Skript erstellen. Nachdem Sie in der Spalte „Auszuführendes Skript“ in die entsprechende Zeile geklickt haben, erhalten Sie dort eine Liste aller verfügbaren Skripte zur Auswahl. Ein einmal konfiguriertes Skriptereignis können Sie durch einen Klick auf das rote Kreuz am Ende der Zeile wieder löschen.

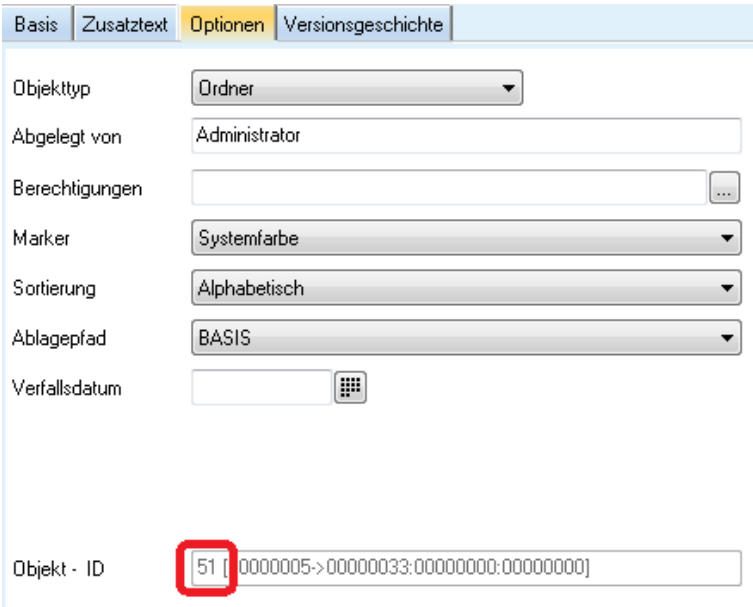


Derart konfiguriert Skriptereignisse müssen für jeden Anwender individuell eingerichtet werden. Es gibt zusätzlich noch einige Skripte, die automatisch ausgeführt werden müssen, ohne dass sie in der Anwenderkonfiguration angemeldet wurden. Diese Skripte beginnen alle mit dem Präfix „ELO_“, eine Liste aller verfügbaren Namen finden Sie im Anhang B.

Startposition im Archiv festlegen

Das erste Beispiel zu einem Skriptereignis ist ein Skript, welches den Anwender beim Programmstart automatisch in einen vorkonfigurierten Ordner führt. Je nachdem, ob man es für alle Anwender oder nur für einzelne Anwender nutzen möchte, kann man das konfigurierte Skriptereignis „Beim Betreten des Archivs“ oder das automatische Skriptereignis „ELO_START“ verwenden. In seiner einfachsten Form gleicht dieses Beispiel dem bereits bekannten GotoId Beispiel. Lediglich das Ziel ist nun fest definiert und wird nicht abgefragt.

Für das Beispiel müssen Sie zuerst den Zielordner im Archiv ermitteln. Sie können diesen dann im Skript entweder über den Zugriffspfad oder über die Objekt-Nummer ansprechen. Die Objektnummer eines Eintrags können Sie im Verschlagwortungsdialog im Reiter „Optionen“ nachschlagen:



Legen Sie ein neues Skript mit dem Namen ELO_START an und tragen dort die folgenden Code Zeilen ein:

```
set ELO = CreateObject("ELO.office")
if ELO.ActionKey = 1 then
  call ELO.GotoId(51)
end if
```

Da das ELO_START Skript beim Starten und Beenden des Archivzugriffs aufgerufen wird und der GotoId Befehl nur beim Start ausgeführt werden soll, muss in dem Skript zuerst geprüft werden, ob ein Start oder ein Ende vorliegt. Das kann anhand des Property ActionKey durchgeführt werden. Beim Start enthält er den Wert 1, beim Beenden den Wert 2.

Statt der festen Objekt-Nummer kann auch der Zugriffspfad definiert werden:

```
set ELO = CreateObject("ELO.office")
if ELO.ActionKey = 1 then
  Id = ELO.LookupIndex("¶Finanzen¶Bestellungen")
  if Id > 0 then
    call ELO.GotoId(Id)
  end if
end if
```

Über den Befehl LookupIndex kann die Objekt-Nummer zum Zielordner oder Dokument eines Ablagepfades ermittelt werden. Ein Ablagepfad in der OLE Schnittstelle wird in ELO immer mit einem Pilcrow Symbol eingeleitet (ALT-0128). Die Kurzbezeichnung dient als Pfadname und die einzelnen Ordnerebenen werden wiederum durch Pilcrow Symbole getrennt.

Wenn der Pfad nicht gefunden wurde, wird eine negative Nummer als Fehlercode zurückgegeben. Nur wenn ein Wert größer als 0 ermittelt wurde, konnte das Zielobjekt gefunden werden. In diesem Fall wird per GotoId zu dem Objekt verzweigt.

Hinweis: Wenn man dem Befehl GotoId einen negativen Wert als Ziel mitgibt, dann springt ELOoffice an diese Stelle und blättert den Zielordner bereits auf, so dass der Anwender direkt eine Selektion vornehmen kann. Wenn das gewünscht ist, kann man den GotoId Befehl so schreiben: call ELO.GotoId(0 – Id).

Anwenderbezogene Startposition

In der nächsten Stufe wird das Beispiel so erweitert, dass für unterschiedliche Anwender unterschiedliche Ziele definiert werden können.

Wenn für ein Aktion eine Vielzahl von Auswahlmöglichkeiten besteht, dann kann man das über verkettete IF Abfragen realisieren:

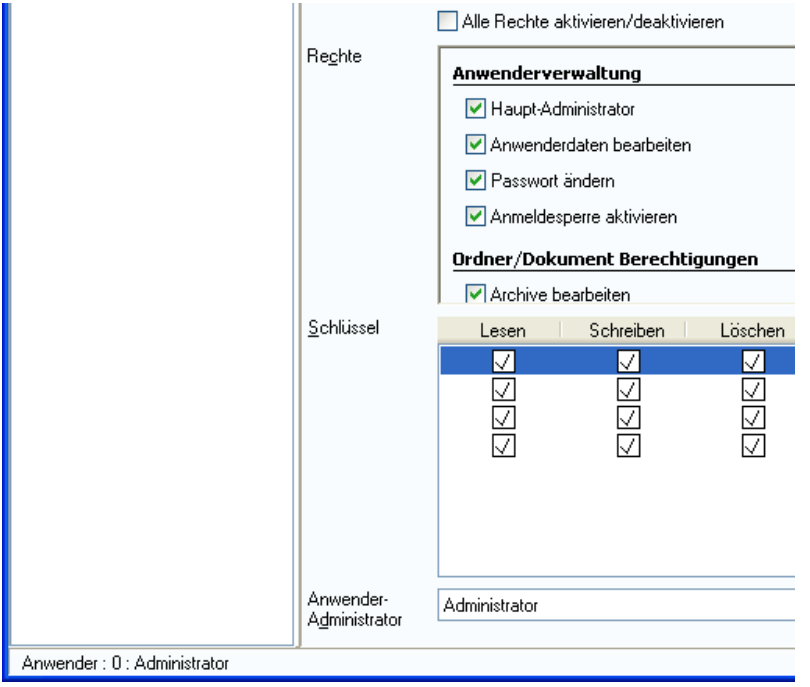
```
Wenn Variante A dann
  Aktion A
Oder Variante B dann
  Aktion B
Oder Variante C dann
  Aktion C
```

Diese Schreibweise führt aber schnell zu unübersichtlichen Konstruktionen. Es gibt aber eine bessere Wahl: das Select – Case Statement. In diesem Fall gibt man einmal den zu prüfenden Wert an, z.B. die Anwendernummer und anschließend eine Liste der Aktionen zu den entsprechenden Werten.

```
set ELO = CreateObject("ELO.office")
if ELO.ActionKey = 1 then
  User = ELO.ActiveUserId
  Select Case User
  case 0
    Id = ELO.LookupIndex("¶Dokumentation¶Beispieldokumente")
  case 1
    Id = ELO.LookupIndex("¶Dokumentation¶Handbücher")
  case 2
    Id = ELO.LookupIndex("¶Dokumentation¶Dokumentenvorlagen")
```

```
  case else
    Id = 0
  end select

  if Id > 0 then
    call ELO.GotoId(0 - Id)
  end if
end if
```

Neu in diesem Skript ist die Abfrage des angemeldeten Anwenders über den Befehl ActiveUserId. Hierüber wird die interne ELO Anwendernummer ermittelt. Sie können diese Nummer in der Anwenderverwaltung in der Statuszeile erkennen, wenn Sie einen Anwender auswählen.

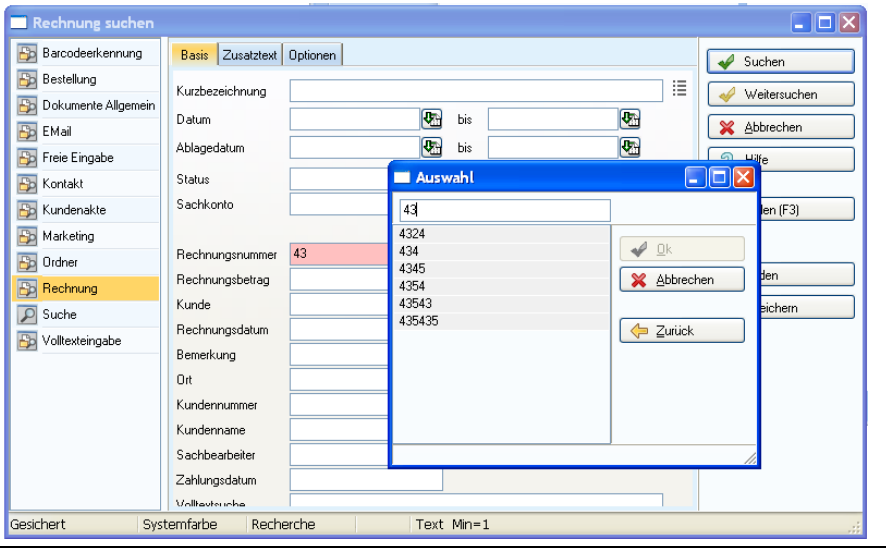


Für jeden Anwender, der eine bestimmte Archivposition erreichen soll, gibt es im Skript einen „case“ Fall. Anwender, die nicht in der Liste aufgeführt werden, erhalten über den „case else“ Fall eine Id 0, für sie wird also kein GotoId ausgeführt.

Verschlagwortung automatisch vorbelegen

ELOoffice bietet verschiedene Möglichkeiten zur Vereinfachung der Eingabe. So können z.B. Indexzeilen automatisch oder per Funktionstaste aus der letzten Verschlagwortung übernommen werden. Manchmal hat man aber Felder, die immer oder fast immer mit dem gleichen Wert gefüllt werden müssen. In diesem Fall wäre es wünschenswert, wenn man in der Maskendefinition einen Vorgabewert hinterlegen könnte. So eine Funktion gibt es in der Version ELOoffice 9.0 nicht, sie kann aber per Skript nachgebildet werden.

Hinweis: Auch ohne Skriptprogrammierung gibt es im ELOoffice viele Wege um wiederkehrende Eingaben zu vereinfachen. Über die Funktionstaste F3 kann die letzte Verschlagwortung wieder eingelesen werden. F7 liefert eine Auswahlliste aller bereits vorhandenen Eingaben die zur aktuellen Teileingabe passen und mittels F8 und F9 kann man die letzte Eingabe einer Zeile automatisch oder manuell wiederholen. Genaueres hierzu finden Sie im ELOoffice Handbuch.



Vorbelegung bei der Archivablage

Zur Implementierung dieser Funktion muss zuerst ein geeignetes Skriptereignis herausgesucht werden. „Nach dem Verschieben aus der Postbox“ oder „Nach dem Neueintrag eines Dokuments in das Archiv“ ist zu spät, denn dann sieht der Anwender den Vorgabewert nicht. Besser ist der Zeitpunkt **„Beim Bearbeiten der Verschlagwortung“**. Hierbei handelt es sich um ein Sammelereignis, d.h. es wird mehrfach von verschiedenen Stellen aufgerufen. Diese Aufrufzeitpunkte sind z.B. beim Starten der Verschlagwortungsanzeige, wenn eine Indexzeile betreten oder verlassen wird und beim Speichern. Die unterschiedlichen Aktionen werden durch unterschiedliche ActionKey Werte markiert und das Skript kann jeweils die notwendigen Befehle ausführen.

Durch folgendes Skript kann man sich einen Überblick verschaffen, wann welche Aufrufe stattfinden. Es wird im Skriptevent „Beim Bearbeiten der Verschlagwortung“ angemeldet und bringt bei jedem Aufruf eine MessageBox mit dem aktuellen ActionKey.

```
Set ELO = CreateObject("ELO.office")
Key = ELO.ActionKey
call ELO.MsgBox("ActionKey: " & Key, "ELO", vbOk Or vbInformation)
```

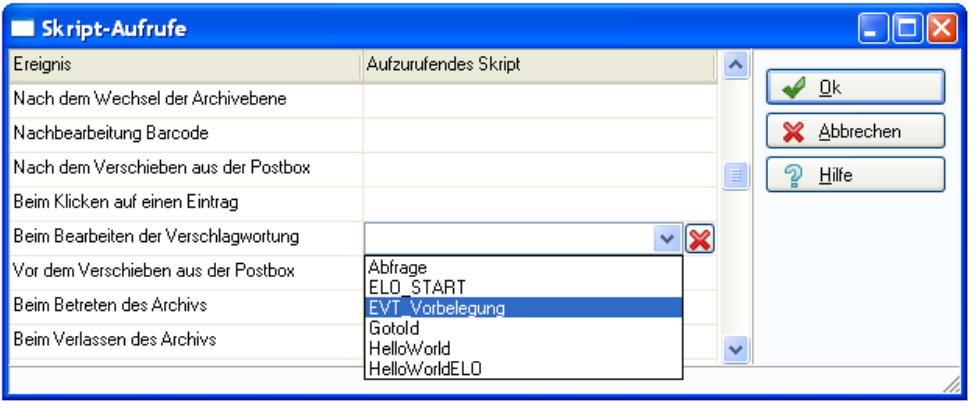
Eine Auflistung der ActionKey Werte und der Skript Ereignisse finden Sie auch in der OLE Automation Anleitung.

Der geeignete Zeitpunkt für unser Beispiel ist der ActionKey Code 20, „Maske betreten“. Der Rahmen für so ein Skript (EVT_Vorbelegung) könnte also so aussehen:

```
set ELO = CreateObject("ELO.office")
if ELO.ActionKey = 20 then
  ' hier folgen die Befehle für die Initialisierung
end if
```

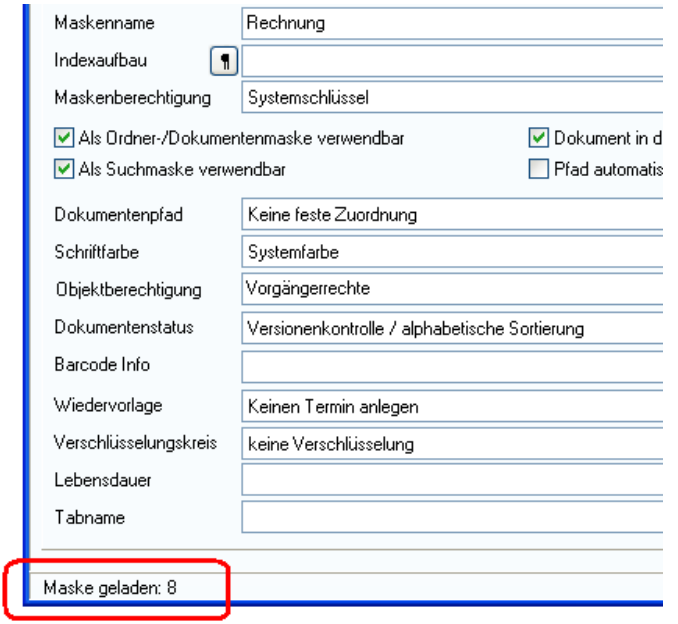
Hinweis: Skripte, die vom Anwender per Schaltfläche aktiviert werden und Skripte, die als Ereignis vom System automatisch aufgerufen werden, können im Normalfall nur für den jeweiligen Aufgabenbereich verwendet werden. Deshalb ist es sinnvoll, diese durch eine passende Benennung zu markieren. Da normale Aktionsskripte ihren Namen in der Schaltfläche sichtbar machen, ist es praktisch, die Ereignisskripte mit einem speziellen Präfix (z.B. EVT_) zu markieren und die Schaltflächenskripte in ihrer Benennung frei zu lassen.

Dieses Skript wird dann als Skript-Aufruf zum Ereignis „Beim Bearbeiten der Verschlagwortung eingetragen.

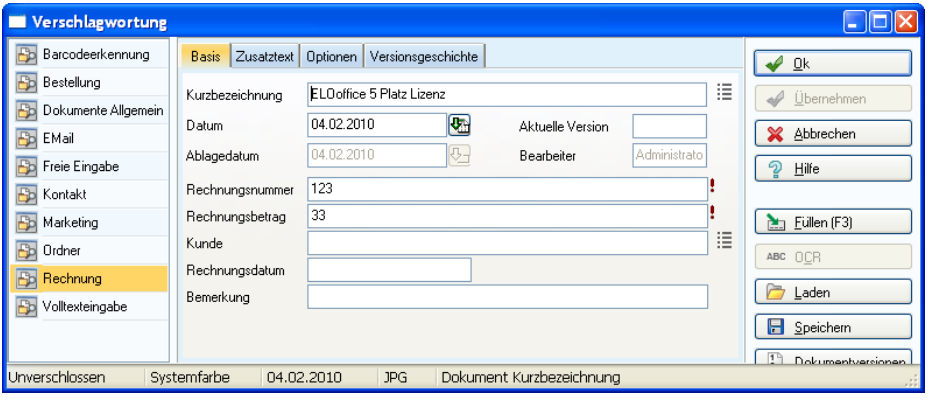


In dieser einfachen Form führt das Skript allerdings noch keine sichtbaren Aktionen aus. Es muss jetzt noch mit Leben gefüllt werden.

Im Demo Archiv gibt es eine Verschlagwortungsmaske „Rechnung“. Die interne Maskennummer dazu kann man in der Statuszeile der Verwaltung sehen.



Das Skript muss in einem ersten Schritt prüfen, ob die aktuelle Verschlagwortung überhaupt ein Rechnungsdokument enthält. In allen anderen Fällen soll es nichts unternehmen. Die aktuell eingestellte Verschlagwortungsmaske kann man über den Befehle GetObjMaskNo ermitteln, dieser Wert muss in unserem Beispiel 8 enthalten. Wenn das der Fall ist, soll die Zeile „Bemerkung“ mit dem Text „Prüfung notwendig“ gefüllt werden.

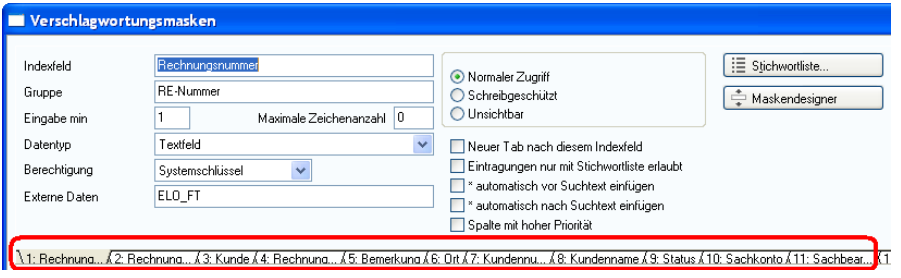


Die erweiterte Version sieht dann so aus:

```
set ELO = CreateObject("ELO.office")
if ELO.ActionKey = 20 then
  if ELO.GetObjMaskNo = 8 then
    call ELO.SetObjAttrib(4, "Prüfung notwendig")
  end if
end if
```

Nach der Prüfung auf die richtige Maskennummer wird als nächstes die Indexzeile 4 durch den Befehl SetObjAttrib mit dem definierten Text gefüllt. Beachten Sie bitte, dass die Zeilennummerierung für die Befehle Get/SetObjAttrib bei 0 beginnt, deshalb hat die fünfte Zeile die Zeilennummer 4 und nicht 5.

Hinweis: Bitte beachten Sie, dass man im Verschlagwortungsdesigner Indexzeilen umsortieren kann. Ein einfaches Abzählen der Zeilen in der Verschlagwortung kann also zu einem falschen Wert führen. Die Indexzeilennummer sollten Sie also immer im Verschlagwortungsmasken Dialog ablesen.



In der aktuellen Form hat das Skript aber noch einen schwerwiegenden Fehler: der Text wird bei jedem Aufruf eingefügt. Wenn zu einem späteren Zeitpunkt die vollständig bearbeitete Rechnung noch mal aufgerufen wird, dann überschreibt das Skript den Text erneut mit „Prüfung notwendig“. Hierzu gibt es zwei mögliche Strategien: man schreibt den Text nur, wenn die Zeile noch leer ist. Oder man schreibt nur dann, wenn das Dokument noch nicht abgelegt wurde. Beide Möglichkeiten haben ihre Vor- und Nachteile.

Die Prüfung auf leere Zeilen kann man nur einsetzen, wenn im Rahmen der normalen Bearbeitung nicht auch Zwischenzustände mit einer leeren Bemerkungszeile auftreten können (diese würde ja automatisch sofort wieder gefüllt werden). Die Prüfung auf eine Neuablage ist dann unpraktisch, wenn die Dokumente von einer zentralen Poststelle erst mal nur „Minimalverschlagwortet“ werden sollen und erst im Laufe der Bearbeitung mit der korrekten Verschlagwortungsmaske versehen werden. Welche der beiden Varianten günstiger ist, muss also immer im Einzelfall entschieden werden. Im Beispiel werden deshalb beide vorgestellt.

```
set ELO = CreateObject("ELO.office")
if ELO.ActionKey = 20 then
  if ELO.GetObjMaskNo = 8 then
    if ELO.GetObjAttrib(4) = "" then
      call ELO.SetObjAttrib(4, "Prüfung notwendig")
    end if
  end if
end if
```

Neu ist hier die Zeile „if ELO.GetObjAttrib(4)…“. Hier wird der aktuelle Inhalt der Indexzeile eingelesen und geprüft, ob diese leer ist. Nur in diesem Fall wird der Inhalt auf „Prüfung notwendig“ gesetzt.

Wenn man nur bei neuen Dokumenten die Indexzeile füllen möchte, dann kann man prüfen, ob der Eintrag bereits eine logische ELO Objektnummer besitzt. Alle archivierten Dokumente haben so eine Nummer. Postboxdokumente besitzen jedoch noch keine. Die logische Objektnummer des aktuellen Eintrags kann man über den Befehl GetEntryId(-2) abfragen. Der Rest des Skriptes ist identisch zur Vorgängerversion.

```
set ELO = CreateObject("ELO.office")
if ELO.ActionKey = 20 then
  if ELO.GetObjMaskNo = 8 then
    if ELO.GetEntryId(-2) = 0 then
      call ELO.SetObjAttrib(4, "Prüfung notwendig")
    end if
  end if
end if
```

Ein kleines Problem gibt es aber immer noch. Wenn das Dokument mit einer anderen Verschlagwortung geladen wird (z.B. „Freie Eingabe“ mit der ObjMaskNo 0), dann wird das Skript zu einem Zeitpunkt ausgeführt, an dem die Maskennummer noch nicht 8 ist. Die Indexzeile wird deshalb nicht gefüllt. Wenn der Anwender nun im Verschlagwortungsdialog auf „Rechnung“ umschaltet, wird nicht noch mal das Ereignis „Verschlagwortung gestartet“ aufgerufen und die Bemerkung bleibt ohne Vorbelegung. Hier muss man also zusätzlich das Ereignis „Verschlagwortungsmaske geändert“ (ActionKey = 23) überwachen, das Skript sieht dann so aus:

```
set ELO = CreateObject("ELO.office")
if ELO.ActionKey = 20 or ELO.ActionKey = 23 then
  if ELO.GetObjMaskNo = 8 then
    if ELO.GetObjAttrib(4) = "" then
    'if ELO.GetEntryId(-2) = 0 then
      call ELO.SetObjAttrib(4, "Prüfung notwendig")
    end if
  end if
end if
```

Hinweis: Beachten Sie bitte, dass Sie über die OLE Schnittstelle die normalen Eingabeprüfungen des Verschlagwortungsdialogs umgehen. Somit ist es also möglich, dass Sie Werte eintragen, die es an dieser Stelle eigentlich nicht geben sollte (z.B. Texte in Datumsfeldern oder freie Eingabewerte in reinen Stichwortlistenfeldern. Das Verhalten ist gewünscht und an einigen Stellen auch notwendig, deshalb liegt es in Ihrer Verantwortung als Skriptentwickler, dafür zu sorgen, dass nur zulässige Werte verwendet werden.

Vorbelegung in der Suchansicht

Für die normale Verschlagwortung ist das oben aufgeführte Skript nun fertig. Was passiert aber, wenn man eine Suche durchführen möchte. Ein kurzer

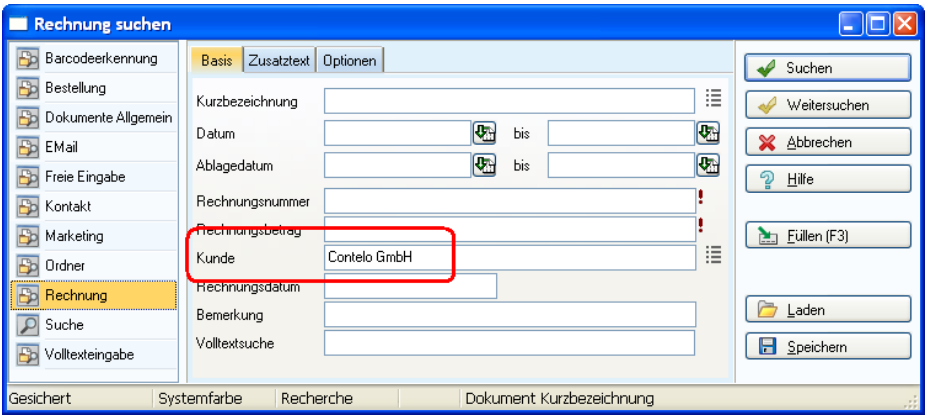
Test zeigt, dass dann weder der ActionKey 20 noch der ActionKey 23 kommt, das Bemerkungsfeld bleibt also leer.

Im Normalfall ist das Verhalten so gewünscht, insbesondere da die Vorbelegung aus dem Beispiel in der Suche äußerst störend wäre. Aber auch in der Suche gibt es durchaus den Bedarf an einer passenden Vorbelegung, nur eben für andere Indexzeilen und mit anderen Werten. Nehmen wir für das Beispiel einmal an, dass wir 95% unseres Geschäfts mit der Contelo GmbH machen. Dann wäre es in der Suche doch praktisch, wenn das Feld Kunde bereits mit „Contelo GmbH“ vorbelegt wäre. In den seltenen Fällen, in denen wir nach einen anderen Kunden suchen, kann man das Feld leicht verändern.

Ein Blick in die Dokumentation zeigt, dass in der Suche im Prinzip die gleichen Ereignisaufrufe kommen, nur haben die ActionKey Werte einen Versatz von Hexadezimal 8000, also 32768 Dezimal. Das erweiterte Skript muss also zusätzlich auf die ActionKeys 32788 und 32791 achten und dort dann die gewünschte Kundenvorbelegung durchführen.

```
set ELO = CreateObject("ELO.office")
ActionKey = ELO.ActionKey
call ELO.DebugOut("ActionKey = " & ActionKey)

' Vorbelegung für die Verschlagwortung
if ActionKey = 20 or ActionKey = 23 then
  if ELO.GetObjMaskNo = 8 then
    if ELO.GetObjAttrib(4) = "" then
      call ELO.SetObjAttrib(4, "Prüfung notwendig")
    end if
  end if
end if

' Vorbelegung für die Suche
if ActionKey = 32788 or ActionKey = 32791 then
  if ELO.GetObjMaskNo = 8 then
    if ELO.GetObjAttrib(4) = "" then
      call ELO.SetObjAttrib(2, "Contelo GmbH")
    end if
  end if
end if
```

Hinweis: In diesem Beispiel wurde die mehrfache Abfrage von ELO.ActionKey gegen die Speicherung in einer lokalen Variablen ausgetauscht. Diese Änderung ist nicht aus Geschwindigkeitsgründen durchgeführt worden sondern um den Report übersichtlicher zu halten. Im Allgemeinen ist die Kommunikation zwischen ELOoffice und den Skripten sehr schnell, so dass hier nur selten eine Optimierung notwendig ist.



Kurzbezeichnung automatisch füllen

Dieses Beispiel erfüllt eine ähnliche Aufgabe wie der Vorgänger. Bei der Rechnungserfassung werden die wichtigen Daten, wie z.B. Kunde, Betrag, Rechnungsnummer, in den Indexfeldern erfasst. In der Kurzbezeichnung soll eine einfache Zusammenfassung dieser Daten die Übersichtlichkeit in der Orderstruktur erhöhen. Es wäre jetzt unnötiger Aufwand, wenn der Anwender diese doppelt eingeben muss, einmal in der Kurzbezeichnung und einmal in den Indexzeilen. Deshalb soll die Kurzbezeichnung automatisch aus den Indexzeilen für den Kundennamen und der Rechnungsnummer gefüllt werden.

Das benötigte Skriptereignis ist das gleiche wie zuvor: „Beim Bearbeiten der Verschlagwortung“. Allerdings wird nun als ActionKey der Zeitpunkt des Speicherns benötigt. Der ActionKey für „Speichern“ lautet 21. Eine erste einfache Version des Skripts könnte so aussehen:

```
set ELO = CreateObject("ELO.office")
ActionKey = ELO.ActionKey
call ELO.DebugOut("ActionKey = " & ActionKey)

if ActionKey = 21 then
  if ELO.GetObjMaskNo = 8 then
    Name = ELO.GetObjAttrib(2)
    Nummer = ELO.GetObjAttrib(0)
    ELO.ObjShort = Name & " : " & Nummer
  end if
end if
```

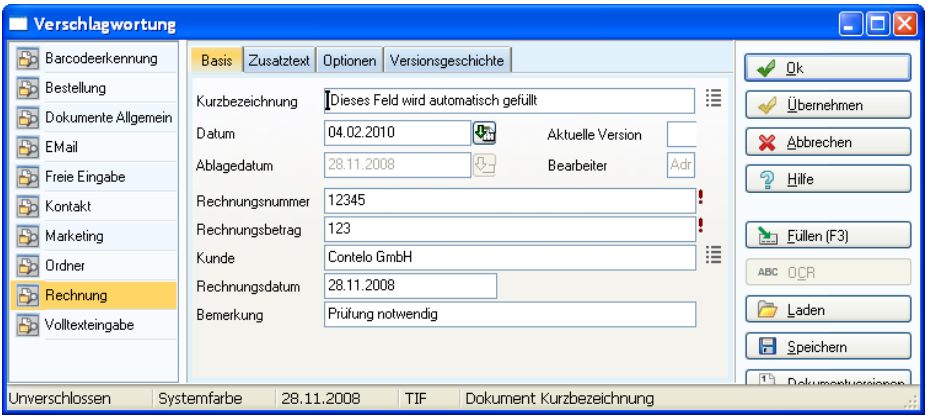
Auch wenn das Skript im Prinzip das macht, was gefordert wurde, hat es aus der Sicht des Anwenders einen erheblichen Schönheitsfehler: er bekommt keine Rückmeldung darüber, was in der Kurzbezeichnung stehen wird. Er muss den Mut haben, die Kurzbezeichnung „verbotenerweise“ leer zu lassen (was eigentlich nicht zulässig ist und ohne Skript zu einer Fehlermeldung beim Speichern führt) und sieht erst beim erneuten Aufruf oder in der Ordnerstruktur, was für eine Bezeichnung eingetragen wurde. Schöner wäre es, wenn man schon während der Verschlagwortung erkennen könnte, was dort eingetragen wird.

Für diese direkte Rückmeldung werden die ActionKey Werte beim Verlassen einer Indexzeile ausgewertet. Wenn der Anwender einen Kundennamen oder eine Kundennummer eingegeben hat, wird die Kurzbezeichnung sofort aktualisiert. Der ActionKey für das Verlassen einer Indexzeile lautet <Zeilennummer> + 2000.

```
set ELO = CreateObject("ELO.office")
ActionKey = ELO.ActionKey
call ELO.DebugOut("ActionKey = " & ActionKey)

if ActionKey = 21 or ActionKey = 2000 or ActionKey = 2002 then
  if ELO.GetObjMaskNo = 8 then
    Name = ELO.GetObjAttrib(2)
    Nummer = ELO.GetObjAttrib(0)
    ELO.ObjShort = Name & " : " & Nummer
  end if
end if
```

Jetzt bleibt noch ein weiteres Problem offen: wenn der Anwender nicht weiß, dass die Kurzbezeichnung automatisch gefüllt wird, gibt er beim Betreten der Verschlagwortungsmaske etwas ein, was dann später verloren geht. Deshalb soll beim Betreten der Kurzbezeichnung dort ein Text hinterlegt werden, der darauf hinweist, dass man hier nichts eingeben soll. Wenn das Feld verlassen wird, soll wieder der richtige Text eingetragen werden. Die ActionKey Werte für das Betreten und Verlassen der Kurzbezeichnung lauten 10 und 11.

```
set ELO = CreateObject("ELO.office")
ActionKey = ELO.ActionKey
call ELO.DebugOut("ActionKey = " & ActionKey)

if ActionKey = 10 then
  if ELO.GetObjMaskNo = 8 then
    ' Kurzbezeichnung betreten: Warnhinweis ausgeben
    ELO.ObjShort = "Dieses Feld wird automatisch gefüllt"
  end if
end if

if ActionKey = 11 or ActionKey = 21 or ActionKey = 2000_
     or ActionKey = 2002 then
  if ELO.GetObjMaskNo = 8 then
    Name = ELO.GetObjAttrib(2)
    Nummer = ELO.GetObjAttrib(0)
    ELO.ObjShort = Name & " : " & Nummer
  end if
end if
```


Verschlagwortung
Barcodeerkennung
Bestellung
Dokumente Allgemein
EMail
Freie Eingabe
Kontakt
Marketing
Ordner
Rechnung
Volltexteingabe
Basis
Zusatztext
Optionen
Versionsgeschichte
Kurzbezeichnung
Dieses Feld wird automatisch gefüllt
Datum
04.02.2010
Aktuelle Version
Ablagedatum
28.11.2008
Bearbeiter
Adr
Rechnungsnummer
12345
Rechnungsbetrag
123
Kunde
Contelo GmbH
Rechnungsdatum
28.11.2008
Bemerkung
Prüfung notwendig
Ok
Übernehmen
Abbrechen
Hilfe
Füllen (F3)
OCR
Laden
Speichern
Unverschlossen
Systemfarbe
28.11.2008
TIF
Dokument Kurzbezeichnung

Programmierfehler vermeiden

Das Beispiel aus dem letzten Kapitel ist über mehrere Stufen schrittweise gewachsen. Deshalb hat es der Sicht des Software Engineering ein paar erhebliche Schwächen angesammelt. Es ist an der Zeit für eine Überarbeitung – heutzutage Refactoring genannt. Dabei geht es nicht darum, bestehende Skripte zu erweitern. Sondern sie sollen so verbessert werden, dass der Code übersichtlicher, besser wartbar und zuverlässiger wird.

Werfen wir zuerst einen Blick auf die Probleme der aktuellen Version:

```
set ELO = CreateObject("ELO.office")
ActionKey = ELO.ActionKey
call ELO.DebugOut("ActionKey = " & ActionKey)

if ActionKey = 10 then
  if ELO.GetObjMaskNo = 8 then
    ' Kurzbezeichnung betreten: Warnhinweis ausgeben
    ELO.ObjShort = "Dieses Feld wird automatisch gefüllt"
  end if
end if

if ActionKey = 11 or ActionKey = 21 or ActionKey = 2000_
    or ActionKey = 2002 then
  if ELO.GetObjMaskNo = 8 then
    Name = ELO.GetObjAttrib(2)
    Nummer = ELO.GetObjAttrib(0)
    ELO.ObjShort = Name & " : " & Nummer
  end if
end if
```

- Gleichartige Codeteile kommen mehrfach vor. So etwas entsteht durch „Copy and Paste“ Programmierung. Programmteile, die an einer Stelle erfolgreich eingesetzt wurden, werden einfach kopiert statt in eine Unterfunktion ausgelagert. Bei späteren Erweiterungen muss man darauf achten, dass alle Verwendungsstellen gefunden und korrekt angepasst werden. Das ist nicht nur aufwändig sondern auch sehr fehlerträchtig.

- Das Programm ist übersät mit „magic numbers“: ActionKey = 2002, GetObjAttriib(2), GetObjMaskNo = 8. Diese Zahlen sind während der Entwicklung schnell in der Dokumentation nachgeschlagen. Aber wer kennt diese noch nach einem halben Jahr? Was war noch mal der ActionKey 2002? Wenn man das Skript verstehen will, muss man alle Werte in der Dokumentation erneut suchen.
- VB Script bietet einen Schutz gegen Tippfehler in Variablennamen mit „option explicit“. Bei einem kleinen Skript mit ein paar Zeilen mag das unnötig erscheinen. Bedenken Sie aber, dass Programme dazu neigen, mit der Zeit immer stärker zu wachsen. Nicht entdeckte Tippfehler in Variablennamen führen später zu schwer erkennbaren Fehlfunktionen und sind deshalb äußerst gefährlich.
- Manche Entwickler vertreten die Ansicht, dass eine Funktion oder Subroutine maximal 7 Codezeilen enthalten sollte. Ob man jetzt genau 7 Zeilen oder 12 oder 20 Zeilen als Grenze ansieht, ist eine Geschmacksfrage und hängt auch von der Situation ab. Wenn eine Funktion aber 50 Zeilen hat, dann ist die Zeit für ein Refactoring gekommen. Auch hier sollten Sie bedenken, dass Funktionen im Laufe der Zeit immer weiter anwachsen, so dass eine geeignete Strukturierung durchaus frühzeitig vorgenommen werden kann.
- Kommentare wurden äußerst sparsam eingesetzt. Einen guten Kommentar zu schreiben ist eine Kunst für sich. Auf keinen Fall sollten Sie einfach nur wiederholen, was im Code schon offensichtlich ist: „anw = anw + 1 'Zähler um eins erhöhen“ ist ein nutzloser Kommentar, der die Zeit des Lesers unnütz verschwendet. Durch eine gute Wahl von Variablennamen kann man zusätzlich eine automatische Kommentierung im Programmcode erreichen: „AnzahlAnwender = AnzahlAnwender + 1“ benötigt keinen weiteren Kommentar.
- Es ist nicht vermerkt, wer für das Skript verantwortlich ist und wofür es eingesetzt werden soll.

Im Folgenden finden Sie eine überarbeitete Version des Skripts. Es ist nun zwar länger geworden, dafür kann man die Funktion aber auch leichter

verstehen und Änderungen vornehmen. In der ursprünglichen Version war die Nummer der Rechnungsmaske über mehrere Stellen im Programmcode verstreut. Wenn das Skript auf eine neue Maske umgestellt werden sollte, mussten alle Stellen gefunden werden. Dabei kann man aber nicht einfach alle numerischen Konstanten 8 auf den neuen Wert umstellen. Man muss für jede Fundstelle prüfen, ob diese 8 für die Rechnungsnummer oder eine andere Funktion steht. In der neuen Version gibt es gleich am Anfang des Skripts eine Konstantendefinition für die Rechnungsmaske, bei einer Änderung muss nur diese eine Stelle angepasst werden. Fehler können dabei nicht passieren.

```
' Skript: EVT_VerschlagwortungBearbeitet
' Autor: Matthias Thiele
' Erstellt: 11.06.2010
' Letzte Änderung: 12.06.2010
'
' Dieses Skript füllt die Kurzbezeichnung automatisch
' mit den Eingaben aus den Feldern Kundennummer und
' Kundenname. Damit der Anwender nicht irrtümlich
' selber Eingaben in der Kurzbezeichnung vornimmt,
' wird das Feld beim Betreten mit einem Warnhinweis
' gefüllt.


option explicit
Dim ELO


' Maskennummer und Maskenfelder
Const InvoiceMaskNo = 8
Const IndexKundennummer = 0
Const IndexKundenname = 2

' ActionKey Werte
Const AK_EnterShort = 10
Const AK_ExitShort = 11
Const AK_MaskChanged = 21
Const AK_EnterCustomerNumber = 2000
Const AK_EnterCustomerName = 2002
```

```
set ELO = CreateObject("ELO.office")
call AutoInsert


sub AutoInsert
  Dim ActionKey, MaskNumber
  ActionKey = ELO.ActionKey
  MaskNumber = ELO.GetObjMaskNo
  call ELO.DebugOut("ActionKey = " & ActionKey)
  if MaskNumber = InvoiceMaskNo then
    call SetHintMessage(ActionKey)
    call SetShortDescription(ActionKey)
  end if
end sub


' Wenn die Kurzbezeichnung betreten wird, dann soll dort ein
' kurzer Text angezeigt werden, der dem Anwender signalisiert,
' dass er hier keine Eingaben vornehmen soll

sub SetHintMessage(ActionKey)
  if ActionKey = AK_EnterShort then
    ELO.ObjShort = "Dieses Feld wird automatisch gefüllt"
  end if
end sub


' Wenn eines der Felder, aus der sich die Kurzbezeichnung
' zusammensetzt verlassen wird, wird diese mit dem automatisch
' generierten Wert aktualisiert.
' Wenn die Kurzbezeichnung selber verlassen wird, wird diese
' ebenfalls aktualisiert damit der Warnhinweis wieder entfernt wird

sub SetShortDescription(ActionKey)
  Dim Name, Nummer
  if ActionKey = AK_ExitShort or ActionKey = AK_MaskChanged _
     or ActionKey = AK_EnterCustomerNumber _
     or ActionKey = AK_EnterCustomerName then
    Name = ELO.GetObjAttrib(IndexKundenname)
    Nummer = ELO.GetObjAttrib(IndexKundennummer)
    if Name <> "" or Nummer <> "" then
      ELO.ObjShort = Name & " : " & Nummer
    else
      ELO.ObjShort = ""
    end if
```

```
  end if
end sub
```

Hinweis: Über den korrekten Aufbau von Variablennamen kann man beliebig lange streiten. Deutsche oder Englische Bezeichner? Groß/Kleinschreibweise? Camel Case (DasIstEinWortImCamelCase)? Unterstriche, ja oder nein? Oder eine Kombination? Egal wofür man sich entscheidet – man sollte es dann konsequent einsetzen. Für den späteren Bearbeiter wird die Einarbeitung dann wesentlich leichter.

Hinweis: White space kostet nichts. Gehen Sie nicht zu sparsam mit Leerzeilen und Leerzeichen um. Eine geeignete Strukturierung in Absätze verhilft nicht nur bei einem Brief zu besserer Lesbarkeit.

Aktuellen Archiveintrag ermitteln

In vielen Skripten ist es notwendig, dass man die aktuelle Archivposition, Auswahl oder die Nummer des gerade aktiven Eintrags ermittelt. Dabei kann es manchmal etwas verwirrend sein, welche Information man tatsächlich benötigt. Deshalb wird im Folgenden der Unterschied erklärt.

Einer der drei aufgeführten Kandidaten ist im Beispiel zuvor schon aufgetreten: die Objektnummer des aktiven Eintrags. Diese Information ist besonders bei Skriptereignissen, wie z.B. während der Verschlagwortung oder beim Verschieben, notwendig. Sie kann über den Aufruf GetEntryId(-2) gelesen werden.

Eine sehr ähnliche Aktion ist die Ermittlung des aktuell selektierten Eintrags, er wird per GetEntryId(-1) gelesen. Auch wenn das in vielen Fällen der gleiche Eintrag sein wird, ist das nicht immer der Fall. Deshalb ist es wichtig, die Unterschiede zu kennen. Nur dann kann man je nach Situation die richtige Auswahl treffen.

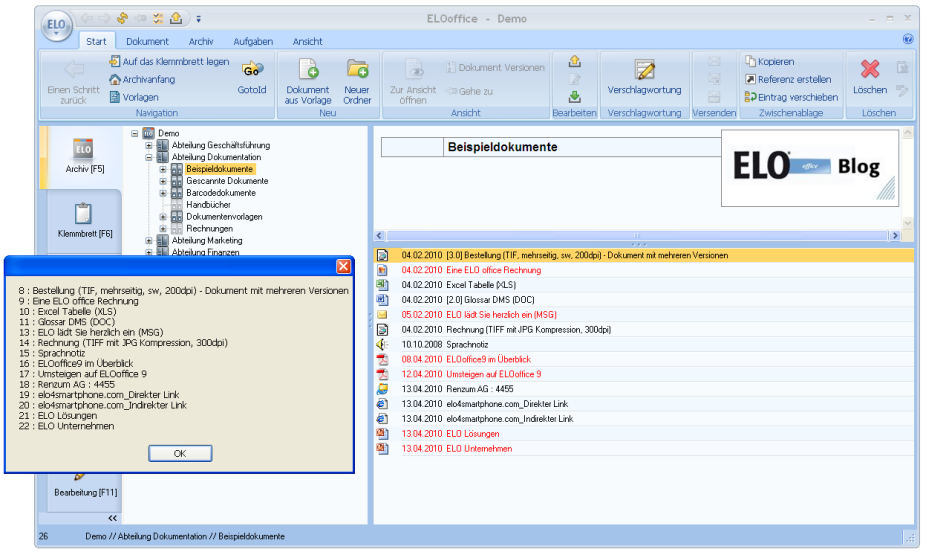
Bei vielen Skriptereignissen, z.B. beim Bearbeiten der Verschlagwortung, ist der gerade selektierte Eintrag aktiv in Bearbeitung. In diesem Fall sind die Objektnummern aus GetEntryId(-1) (selektierter Eintrag) und GetEntryId(-2) (aktiver Eintrag der OLE Automation Schnittstelle) identisch. Das kann sich aber während der Skriptbearbeitung ändern. Wenn Sie in einem Skript einen neuen Ordner erzeugen, dann bleibt der selektierte Eintrag mit der ursprünglichen Nummer erhalten. Der aktive Eintrag ist nun aber der neue Ordner.

Neben dem selektierten Eintrag und dem aktiven Eintrag benötigt man oft Zugriff auf die Objektnummern aller Einträge in einer Liste. Auch dieser Zugriff wird über GetEntryId realisiert. In diesem Fall wird als Parameter die Zeilennummer mitgegeben, die oberste Zeile hat die Nummer 0. Zur Auflistung aller Zeilen kann man nicht vorab die Anzahl der Einträge abfragen. Das Skript muss bei 0 beginnend aufsteigend jeweils die nächste Eintragsnummer abfragen. Sobald die Liste am Ende angekommen ist, wird

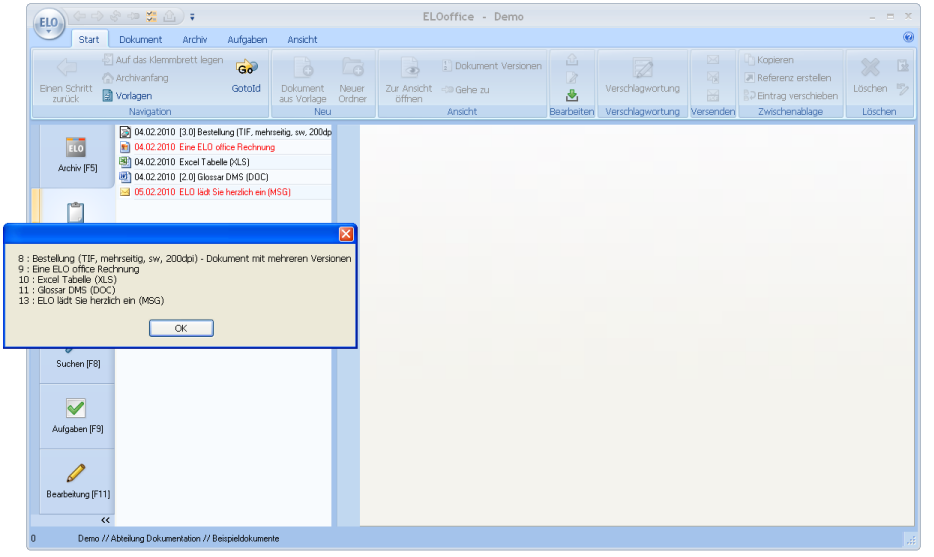
eine 0 zurückgemeldet. An dieser Stelle muss die Auflistung abgebrochen werden.

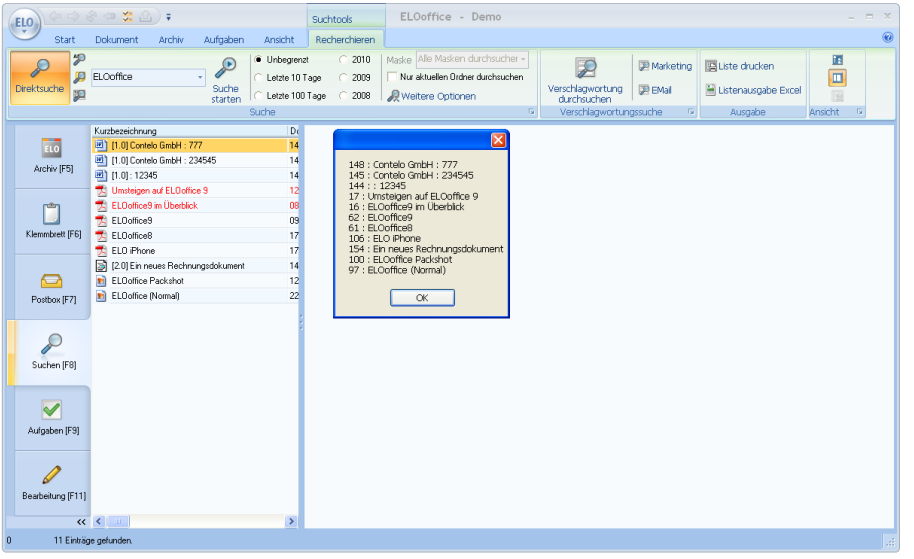
```
option explicit

Dim ELO, Line, GoOn, Id, Msg
Set ELO = CreateObject("ELO.office")

Line = 0
GoOn = TRUE
While GoOn
  Id = ELO.GetEntryId(Line)
  If Id < 1 Then
    GoOn = FALSE
  Else
    Line = Line + 1
    Msg = Msg & Id & " : " & ELO.GetEntryName(Id) & vbCrLf
  End If
Wend

MsgBox Msg
```

In der Archivansicht werden alle Untereinträge der aktuellen Auswahl angezeigt.



Im Klemmbrett und in der Suche werden alle Einträge der Liste angezeigt.





Hinweis: In der Aufgabenansicht kann das Skript nicht verwendet werden. Dort liefert GetEntryId nicht die logische Objektnummer des Dokuments sondern die Nummer des Termins. In der Postbox gibt es keine Objektnummern, so dass es hier auch nicht geht. In der Bearbeitung liefert das Skript zwar die richtigen Nummern. Dort muss man aber sehr vorsichtig mit Veränderungen sein. Wenn man hier die Dokumentensperre durch eine Skriptaktion entfernt, kann der Anwender anschließend seine Veränderungen nicht mehr einchecken.

Ordner oder Dokumente erzeugen

Häufig kommt im Rahmen der Skriptprogrammierung die Anforderung auf, automatisiert einen Ordner oder ein Dokument zu erzeugen. Dieser Vorgang wird in drei Schritten abgearbeitet:

1. Mittels PrepareObjectEx wird eine leere Verschlagwortung erzeugt.
2. Über Properties und Zugriffsfunktionen, wie z.B. ObjShort oder SetObjAttrib() wird die Verschlagwortung mit Daten gefüllt. Weiterhin muss das Ablageziel festgelegt werden.
3. Ein abschließendes UpdateObject erzeugt dann das logische ELO Objekt in der Datenbank.

Neuen Ordner erstellen

Das folgende Skript soll die notwendigen Schritte zur Erzeugung eines neuen Ordners auf der Schrankebene aufzeigen.

```
Set ELO = CreateObject("ELO.office")
const Level = 1
const FolderMaskNo = 1
const ParentId = 1
const CreateNewEntry = 0

' Leeres Objekt vorbereiten
call ELO.PrepareObjectEx(CreateNewEntry, Level, FolderMaskNo)

' Objekt mit Daten füllen
ELO.ObjShort = "Ein neuer Schrank"
ELO.ObjIndex = "#" & ParentId
call ELO.SetObjAttrib(0, "Test")

' In der Datenbank abspeichern
if ELO.UpdateObject() < 0 then
  call ELO.MsgBox("Es ist ein Fehler beim Speichern aufgetreten.",
"ELO", vbOkOnly)
end if
```

Die Konstante „Level“ bestimmt den Ordnertyp (Schrank = 1, Ordner = 2, Register = 3. …). Da ein Schrank erzeugt werden soll, wird dieser Wert auf 1 eingestellt. Die Maske „Ordner“ hat im Demo Archiv und in fast jedem anderen neueren Archiv die Nummer 1. Im Zweifelsfall können Sie diesen Wert in Ihrem Archiv in der Verschlagwortungsmaskenverwaltung nachsehen. Die ParentId gibt an, unter welchem Ordner der neue Eintrag erzeugt werden soll. Der Startpunkt „Archiv“ hat immer die interne Objektnummer 1. Die beiden ersten Werte werden bereits für die Erzeugung des leeren Objekts benötigt: call ELO.PrepareObjectEx(0, Level, MaskNo).

Hinweis: Hier zeigt sich schon der Vorteil bei der Verwendung von Konstanten mit sprechenden Namen gegenüber der direkten Verwendung einer Zahl. Eine Zeile call ELO.PrepareObjectEx(0, 1, 1) wäre wesentlich schlechter zu verstehen als ELO.PrepareObjectEx(CreateNewEntry, Level, FolderMaskNo).

Auf die Kurzbezeichnung kann über das Property ObjShort zugegriffen werden. Auch das Ablageziel wird über ein Property bestimmt: ObjIndex. Hier wird entweder ein Zugriffspfad (¶Schrankname¶Ordnername¶Registername) oder eine Nummer mit einem führenden # eingegeben („#1“ ist der Archiv Startpunkt). Einzelne Indexzeilen werden per SetObjAttrib(<Zeilennummer>, <Text>) gefüllt. Beachten Sie bitte, dass die Zählung der Zeilennummern mit 0 beginnt.

Die Subroutine UpdateObject hat keine Parameter. Sie darf erst aufgerufen werden, wenn alle notwendigen Verschlagwortungsdaten geschrieben wurden. Es muss mindestens eine Kurzbezeichnung (ObjShort) und das Ziel (ObjIndex) eingegeben werden. Falls die verwendete Maske Zeilen mit Zwangseingaben besitzt, liegt es in Ihrer Verantwortung als Skriptentwickler, diese zu füllen.

Neues Dokument erstellen

Zur Erstellung eines neuen Dokuments gibt es verschiedene Wege. Wenn das Dokument in der ELO Postbox vorliegt, dann ist der Befehl „MoveToArchiveEx“ geeignet. Liegt das Dokument im Windows Dateisystem und außerhalb der ELO Postbox, dann kann man eine ähnliche Vorgehensweise wie der Anlage eines Ordners beschreiten.

1. Mittels PrepareObjectEx wird eine leere Verschlagwortung erzeugt.
2. Über Properties und Zugriffsfunktionen, wie z.B. ObjShort oder SetObjAttrib() wird die Verschlagwortung mit Daten gefüllt.
3. UpdateObject erzeugt dann das logische ELO Objekt in der Datenbank. In diesem Moment ist das dann noch ein Dokument ohne Dokumentendatei.
4. Zum Abschluss wird mit UpdateDocument noch eine Dokumentendatei in das logische Dokument eingefügt.

Hinweis: das folgende Skript erwartet eine Datei mit dem Namen „Rechnung.TIF“ im Verzeichnis „C:\temp“. Diese muss vor dem Start des Skripts hinterlegt werden, andernfalls erhalten Sie eine Fehlermeldung, dass ein Fehler beim Speichern aufgetreten sei. Sie können auch eine andere Datei an einer anderen Stelle verwenden, dazu müssen Sie lediglich den Zugriffspfad im Befehl UpdateDocument entsprechend anpassen.

```
' Skript: Neues Dokument
' Autor: Matthias Thiele
' Erstellt: 14.06.2010
' Letzte Änderung: 14.06.2010
'
' Dieses Programm legt ein neues Dokument
' in der Schrankebene an. Die Dokumentendatei
' wird aus C:\Temp\Rechnung.TIF gelesen.


option explicit
```

```
Dim ELO, DocId, Result
Set ELO = CreateObject("ELO.office")
const Level = 254
const MaskNo = 8
const ParentId = 1
const CreateNewEntry = 0
const ActiveObjectId = -2

call ELO.PrepareObjectEx(CreateNewEntry, Level, MaskNo)
ELO.ObjShort = "Ein neues Rechnungsdokument"
ELO.ObjIndex = "#" & ParentId
call ELO.SetObjAttrib(0, "12345")
call ELO.SetObjAttrib(1, "145,13")
call ELO.SetObjAttrib(2, "Contelo GmbH")

if ELO.UpdateObject() < 0 then
  call ELO.MsgBox("Es ist ein Fehler beim Speichern aufgetreten.",
"ELO", vbOkOnly)
else
  DocId=Elo.GetEntryId(ActiveObjectId)
  Result = Elo.UpdateDocument( DocId, 0, "C:\Temp\Rechnung.TIF" )
  if Result < 0 then
    call ELO.MsgBox("Es ist ein Fehler beim Speichern aufgetreten: " _
                & Result, "ELO", vbOkOnly)
  end if
end if
```

Funktionen für die Verschlagwortung

In den Beispielen weiter oben sind schon ein paar Properties für die Verschlagwortung aufgeführt worden. Im Folgenden sollen nun die wichtigsten verfügbaren Funktionen und Properties aufgelistet und besprochen werden.

Basisdaten aus der Verschlagwortung: diese Werte befinden sich im Verschlagwortungsdialog auf der Basis-, Zusatztext- und der Optionenseite.

ObjShort	Kurzbezeichnung – Dieser Wert muss zwingend gesetzt werden. Wenn ein Objekt ohne Kurzbezeichnung gespeichert wird, kann es im Archiv nicht richtig angezeigt werden und führt an einigen Stellen zu Problemen bei der Verarbeitung.
ObjIDate	Ablagedatum (Internes Datum) – Das Ablagedatum wird beim Speichern automatisch ermittelt und muss nicht gesetzt werden.
ObjXDate	Dokumentendatum (eXternes Datum) – Das Dokumentendatum kann optional gesetzt werden. Anders als der Name vermuten lässt, darf es auch für Ordner gesetzt werden. Beachten Sie unbedingt, dass dieses Datum nur Tagesgenau (z.B. auf den 13.04.2010) und nicht Minutengenau (13.04.2010 14:20) gesetzt werden darf.
ObjSReg	Versionsnummer – Wenn die Anzeige der Versionsinformation im Verschlagwortungsdialog aktiviert ist, dann wird dieser Wert angezeigt. Er wird aber bei jedem CheckIn Vorgang von der neuen Versionsnummer überschrieben.

ObjMemo **ObjMemoInfo**	Zusatztext – der Zusatztext besteht aus zwei Teilen. Den sichtbaren Teil können Sie mit ObjMemo bearbeiten. Er kann vom Anwender im Verschlagwortungsdialog eingesehen und verändert werden. Der zweite, unsichtbare Teil wird mit ObjMemoInfo gesetzt. Er kann nur per Skript gelesen oder geschrieben werden, für den Anwender ist er weder sichtbar noch änderbar. Beide Teile zusammen können maximal 30000 Zeichen enthalten.
ObjTypeEx	Objekttyp – bei Ordnern „Schrank – Ordner – Register …“ (Wert 1 bis 32) und bei Dokumenten „Dokument – Word – Excel …“ (Wert 254 bis 285).
ObjOwner	Eigentümer des Eintrags – dieser Wert ist für den Anwender im Feld „Abgelegt von“ sichtbar. In der OLE Schnittstelle wird die Anwendernummer gelesen und geschrieben, nicht der Name.
ObjAcl	Zugriffsberechtigungen auf diesen Eintrag
DocKind	Marker – Farbmarkierung für Ordner oder Dokumente. Dieser Wert enthält keine direkte Farbe sondern eine Nummer als Index in die Liste der definierten Farbmarker.
ObjFlags	Verschiedene Einstellungen, z.B. Sortierung, Versionskontrolle, Verschlüsselung, Volltextkennzeichen
ObjVDate	Verfallsdatum – Verfallsdatum des Dokuments.

Zugriff auf die Werte der Indexzeilen

GetObjAttrib	Liest den Wert einer Indexzeile aus. Die Nummer der Indexzeile wird als Parameter übergeben, die Nummerierung beginnt mit 0 für die erste Zeile.
SetObjAttrib	Setzt den Wert einer Indexzeile. Beachten Sie bitte, dass die OLE Schnittstelle keine Typprüfung vornimmt. Wenn eine Indexzeile als ISO-Datumsfeld definiert wurde, können hier trotzdem Texte eintragen. Diese führen dann später aber zu einer unsinnigen Anzeige im Verschlagwortungsdialog.
GetObjAttribMax **GetObjAttribMin**	Gibt die maximale und minimale Eingabelänge zu einer Indexzeile aus. Diese Information können Sie zur Prüfung des Textes vor einem SetObjAttrib Aufruf verwenden. Die OLE Schnittstelle kontrolliert nicht die Einhaltung dieser Vorgaben.
GetObjAttribType	Gibt den Typ der Indexzeile zurück. Diese Informationen können Sie zur Prüfung der Formatierung und erlaubten Werte eines Textes vor einem SetObjAttrib Aufruf verwenden. Die OLE Schnittstelle prüft diese Werte nicht ab.
GetObjAttribKey	Gibt den Gruppennamen einer Indexzeile zurück. Falls Sie mit einem Skript bestimmte Werte (z.B. die Kundennummer KDNR) in unterschiedlichen Masken füllen müssen und sich diese in unterschiedlichen Indexzeilen befinden, können Sie diese auch über den GetObjAttribKey Wert suchen und ermitteln.

Weitere Properties mit internen Daten

ObjGuid	Eindeutige archivübergreifende Kennzeichnung – Im Gegensatz zur Objektnummer, die sich Export/ Importvorgänge ändern kann, bleibt diese Nummer im Normalfall immer gleich. Für die Integration/ Verlinkung

	mit anderen Programmen ist diese deshalb vorzuziehen. Die GUID ist keine einfache Zahl, sondern ein ca. 40 stelliger Text mit einer Windows GUID. Mit dem Befehl GetObjFromGuid kann eine GUID bei Bedarf in eine normale ELO Objektnummer übersetzt werden.
ObjMainParent	Vorgängerordner des aktuellen Elements. Im Normalfall sollte man nur lesend zugreifen. Wichtig: Wenn Sie ein Dokument in einen anderen Ordner verschieben wollen, dann reicht es nicht aus, diesen Wert zu ändern. Für diese Aktion benötigen Sie den Befehl InsertRef.
ObjStatus	Löschstatus – Gibt an, ob ein Eintrag als gelöscht markiert wurde. 0 = normaler Zustand, -1 = gelöscht markiert.

Indexfeld suchen

Wenn ein Skript für mehrere unterschiedliche Ablagemasken verwendet werden soll, dann ist es nicht immer möglich, die Indexzeilen fest anhand einer Nummer zu lesen oder zu schreiben. Eine Indexzeile KDNR (für „Kundennummer“) wird in der Rechnungsmaske möglicherweise an anderer Stelle stehen als in der Maske Kundenordner. In diesem Fall kann man alle Indexzeilen durchlaufen und den richtigen Eintrag anhand des Gruppennamens oder des Anzeigenamens auswählen.

Verschlagwortung aus Vorgängerordner übernehmen

Diese Technik soll anhand eines Beispiels demonstriert werden. Hierfür legen Sie sich im Archiv eine Verschlagwortungsmaske „Kundenakte“ mit den Indexzeilen KDNR (Kundennummer), KDNAM (Kundenname) und ORT (Wohnort) an.

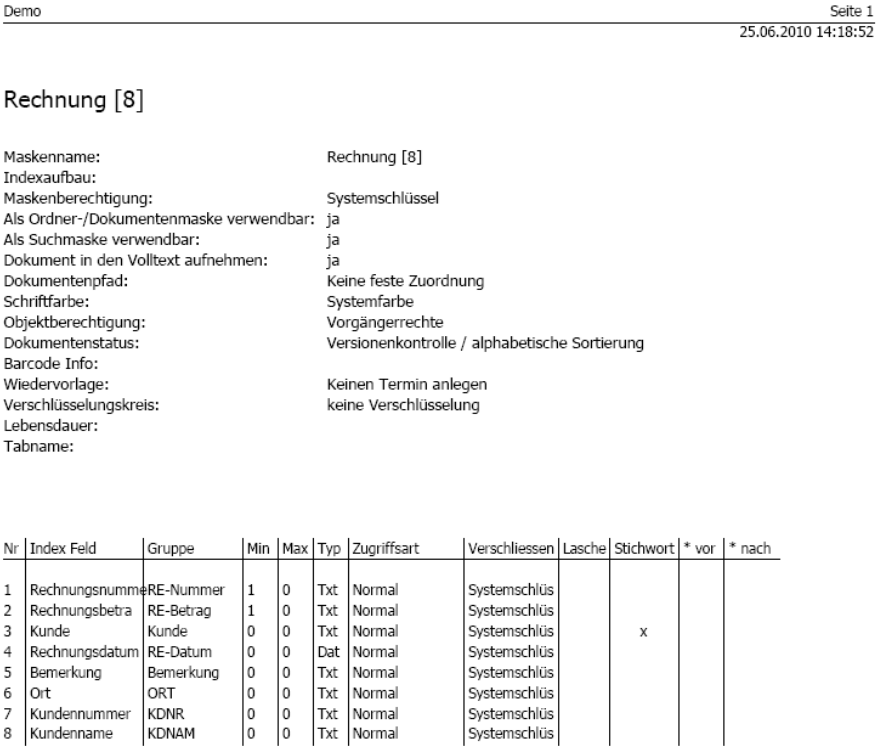
Demo Seite 1
25.06.2010 14:13:02

Kundenakte [11]

Maskenname:	Kundenakte [11]
Indexaufbau:	
Maskenberechtigung:	Systemschlüssel
Als Ordner-/Dokumentenmaske verwendbar:	ja
Als Suchmaske verwendbar:	ja
Dokument in den Volltext aufnehmen:	nein
Dokumentenpfad:	Keine feste Zuordnung
Schriftfarbe:	Systemfarbe
Objektberechtigung:	Vorgängerrechte
Dokumentenstatus:	Versionenkontrolle / alphabetische Sortierung
Barcode Info:	
Wiedervorlage:	Keinen Termin anlegen
Verschlüsselungskreis:	keine Verschlüsselung
Lebensdauer:	
Tabname:	

Nr	Index Feld	Gruppe	Min	Max	Typ	Zugriffsart	Verschliessen	Lasche	Stichwort	* vor	* nach
1	Kundennummer	KDNR	0	0	Txt	Normal	Systemschlüs				
2	Kundenname	KDNAM	0	0	Txt	Normal	Systemschlüs				
3	Ort	ORT	0	0	Txt	Normal	Systemschlüs				

Anschließend laden Sie die Maske „Rechnung“ und ergänzen diese um die gleichen Felder (in beliebiger Reihenfolge).



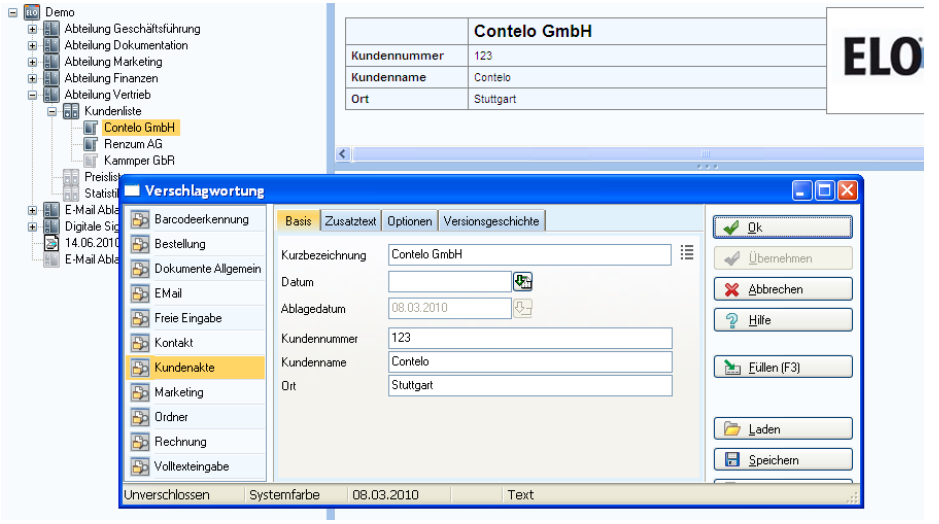
Demo Seite 1
25.06.2010 14:18:52

Rechnung [8]

Maskenname:	Rechnung [8]
Indexaufbau:	
Maskenberechtigung:	Systemschlüssel
Als Ordner-/Dokumentenmaske verwendbar:	ja
Als Suchmaske verwendbar:	ja
Dokument in den Volltext aufnehmen:	ja
Dokumentenpfad:	Keine feste Zuordnung
Schriftfarbe:	Systemfarbe
Objektberechtigung:	Vorgängerrechte
Dokumentenstatus:	Versionenkontrolle / alphabetische Sortierung
Barcode Info:	
Wiedervorlage:	Keinen Termin anlegen
Verschlüsselungskreis:	keine Verschlüsselung
Lebensdauer:	
Tabname:	

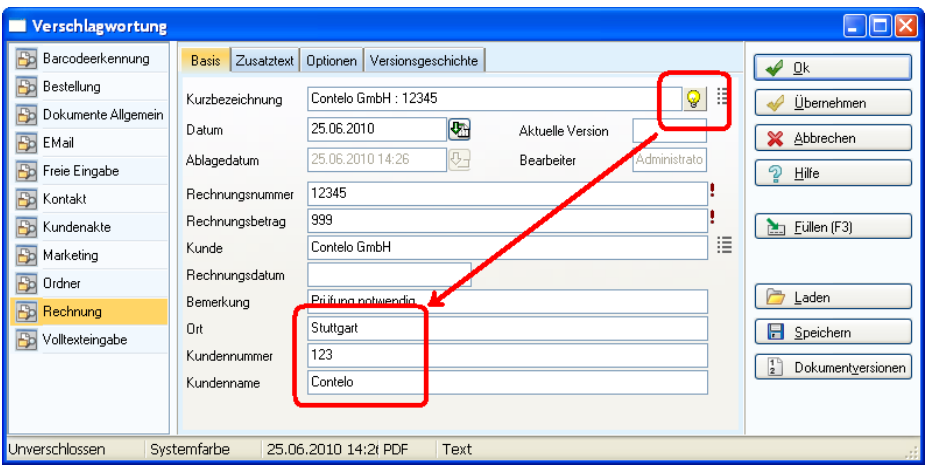
Nr	Index Feld	Gruppe	Min	Max	Typ	Zugriffsart	Verschliessen	Lasche	Stichwort	* vor	* nach
1	Rechnungsnumme	RE-Nummer	1	0	Txt	Normal	Systemschlüs				
2	Rechnungsbetra	RE-Betrag	1	0	Txt	Normal	Systemschlüs				
3	Kunde	Kunde	0	0	Txt	Normal	Systemschlüs		x		
4	Rechnungsdatum	RE-Datum	0	0	Dat	Normal	Systemschlüs				
5	Bemerkung	Bemerkung	0	0	Txt	Normal	Systemschlüs				
6	Ort	ORT	0	0	Txt	Normal	Systemschlüs				
7	Kundennummer	KDNR	0	0	Txt	Normal	Systemschlüs				
8	Kundenname	KDNAM	0	0	Txt	Normal	Systemschlüs				

Im Schrank „Abteilung Vertrieb“, Ordner „Kundenliste“ finden Sie bereits ein paar Kunden im Demo Archiv vor. Diese sind aber mit der Verschlagwortung „Ordner“ abgelegt. Ändern Sie das in „Kundenakte“ und füllen Sie die Indexzeilen aus.



So eine Stammdatenübernahme könnte man automatisch beim Speichern oder Verschieben durchführen. In diesem Beispiel soll aber eine weitere Möglichkeit aufgezeigt werden: durch eine skriptdefinierte Schaltfläche im Verschlagwortungsdialog.

Um eigene Schaltflächen im Verschlagwortungsdialog anzuzeigen, muss ein Ereignisskript „Beim Bearbeiten der Verschlagwortung“ angemeldet werden. Dieses meldet dann beim Start der Verschlagwortung die zusätzlichen Schaltflächen an.



Beim Klick auf die Schaltfläche wird dann die gewünschte Skriptaktion ausgeführt. Es liest die Indexwerte des Vorgängereintrags und durchsucht die aktuelle Verschlagwortung auf Indexzeilen gleichen Namens (im Beispiel nach Gruppennamen und nicht Anzeigenamen). Bei gleichen Indexzeilen wird dann der Wert des Vorgängers in die aktuelle Verschlagwortung übernommen.

Damit die zusätzliche Schaltfläche eingeblendet wird, muss beim ActionKey Aufruf 24 eine Liste aller Indexzeilen angegeben werden, die einen zusätzlichen Button erhalten sollen. Die erste Position ist für die Kurzbezeichnung, die zweite für die Volltexteingabe, danach folgen zwei reservierte Positionen (immer 00 angeben) und anschließend für jede Indexzeile eine 1 oder 0 (1 = Anzeigen, 0 = Nicht Anzeigen. 1000001 zeigt eine Schaltfläche in der Kurzbezeichnung und in der dritten Indexzeile an). Diese Liste wird im Property TextParam übergeben.

```
Select Case ActionKey
  Case AK_QueryButtons
    ELO.TextParam = "1"
```

Wenn der Anwender diese Schaltfläche aktiviert, wird das Ereignis mit dem ActionKey 3999 aufgerufen (Im Skript Beispiel über die Konstante AK_ShortDescClicked definiert). In diesem Fall wird dann der Kopiervorgang gestartet.

```
Select Case ActionKey
  ...

  Case AK_ShortDescClicked
    Call CopyIndexValues()

End Select

Sub CopyIndexValues()
  Call ReadParentValues()
  Call InsertValues()
End Sub
```

Der Kopiervorgang läuft in zwei Stufen ab. Zuerst wird die Verschlagwortung des Vorgängers eingelesen. Dabei ist aber zu beachten, dass dieser Vorgang nicht die aktuelle Verschlagwortung zerstören darf. Diese wird deshalb zuerst mittels SaveObject(1) gesichert und am Ende des Einlesevorgangs mit SaveObject(2) wiederhergestellt.

```
Sub ReadParentValues()
  Dim ParentId, i

  ParentId = ELO.ObjMainParent
  If ParentId > 1 Then
    Call ELO.SaveObject(1)

    Call ELO.PrepareObjectEx(ParentId, 0, 0)
    For i = 0 To MaxIndexLines
      ParentGroups(i) = ELO.GetObjAttribKey(i)
      ParentValues(i) = ELO.GetObjAttrib(i)
    Next

    Call ELO.SaveObject(2)
  End If
End Sub
```

Hinweis: Mittel SaveObject kann man den aktuellen Zustand des Verschlagwortungsobjekts speichern und später wiederherstellen. Das funktioniert aber nur in einer Ebene und kann nicht mehrfach ineinander gestapelt werden. Insbesondere beim Aufruf von Unterfunktionen ist das zu beachten, da man nicht leicht erkennen kann, ob SaveObject bereits vorher verwendet wurde oder nicht.

Im zweiten Teil läuft das Skript über alle Indexwerte des Vorgängers und vergleicht diese mit allen Indexwerten des aktuellen Eintrags. Fall sich eine Namensgleichheit findet, wird der Wert kopiert.

```
Sub InsertValues()
  Dim i, j

  For i = 0 To MaxIndexLines
    If ParentGroups(i) <> "" Then
```

```
      For j = 0 To MaxIndexLines
        If ELO.GetObjAttribKey(j) = ParentGroups(i) Then
          call ELO.SetObjAttrib(j, ParentValues(i))
        End If
      Next
    End If
  Next
End Sub
```

Das vollständige Skript sieht dann so aus

```
' Skript: Stammdatenübernahme
' Autor: Matthias Thiele
' Erstellt: 25.06.2010
' Letzte Änderung: 25.06.2010
'
' Dieses Skript kopiert aus dem übergeordneten
' Ordner alle Indexzeilen die den gleichen
' Gruppennamen in der Verschlagwortung haben.


option explicit

Const AK_QueryButtons = 24
Const AK_ShortDescClicked = 3999
Const MaxIndexLines = 50

Dim ELO, ActionKey, ParentGroups(), ParentValues()
Redim ParentGroups(MaxIndexLines)
Redim ParentValues(MaxIndexLines)

Set ELO = CreateObject("ELO.office")
ActionKey = ELO.ActionKey
Select Case ActionKey
  Case AK_QueryButtons
    ELO.TextParam = "1"

  Case AK_ShortDescClicked
    Call CopyIndexValues()

End Select


Sub CopyIndexValues()
```

```
  Call ReadParentValues()
  Call InsertValues()
End Sub

Sub ReadParentValues()
  Dim ParentId, i

  ParentId = ELO.ObjMainParent
  If ParentId > 1 Then
    Call ELO.SaveObject(1)

    Call ELO.PrepareObjectEx(ParentId, 0, 0)
    For i = 0 To MaxIndexLines
      ParentGroups(i) = ELO.GetObjAttribKey(i)
      ParentValues(i) = ELO.GetObjAttrib(i)
    Next

    Call ELO.SaveObject(2)
  End If
End Sub

Sub InsertValues()
  Dim i, j

  For i = 0 To MaxIndexLines
    If ParentGroups(i) <> "" Then
      For j = 0 To MaxIndexLines
        If ELO.GetObjAttribKey(j) = ParentGroups(i) Then
          call ELO.SetObjAttrib(j, ParentValues(i))
        End If
      Next
    End If
  Next
End Sub
```

Get/ SetObjAttribByName

Da es häufiger vorkommt, dass man eine Indexzeile über den Gruppennamen und nicht über die Zeilennummer Lesen oder Schreiben muss, sollen im Folgenden zwei Funktionen vorgestellt werden, die diese Funktionen anbieten.

GetObjAttribByName – liest den aktuellen Wert aus einem Indexfeld

SetObjAttribByName – schreibt einen Wert in ein bestimmtes Indexfeld

Der Rahmen für beide Funktionen ist sehr ähnlich. Es wird in einer Schleife über alle Indexfelder der Eintrag mit einem bestimmten Gruppennamen gesucht. Wenn dieser gefunden wurde, wird das Feld gelesen oder geschrieben.

```
For i = 0 to 49
  If ELO.GetObjAttribKey(i) = AttribName Then
    ' Hier wird nun gelesen oder geschrieben
    Exit For
  End If
Next
```

Sobald das gewünschte Feld gefunden wurde, kann die Schleife abgebrochen werden. In dieser Form hat die Funktion aber noch einen schwerwiegenden Nachteil: man muss nicht nur den Gruppennamen kennen, man muss zusätzlich auch noch die Schreibweise exakt einhalten. Wenn LocalName = „Kdnr“ und der Gruppenname = „KDNR“ ist, dann wird dieser Eintrag nicht gefunden werden. Deshalb wird die Schleife noch um eine Konvertierung in Kleinschreibweise ergänzt:

```
LocalName = LCase(AttribName)
For i = 0 to 49
  If LCase(ELO.GetObjAttribKey(i)) = LocalName Then
    ' Hier wird nun gelesen oder geschrieben
    Exit For
  End If
Next
```

Hinweis: Prinzipiell könnte man auch eine Konvertierung in Großschreibweise durchführen. Da es aber Zeichen gibt, die nur in einer Kleinschreibweise existieren (z.B. ß), ist diese Variante besser.

Zur Demonstration der Verwendung dieser Funktionen wurde noch ein kleines Rahmenprogramm vorangestellt. Dort wird der Schrank mit der ELO Objektnummer 2 gelesen (zumindest im Demo Archiv ist das ein Schrank). Anschließend wird dann die Indexzeile mit dem Gruppennamen „ELOINDEX“ gelesen. Da der Maskentyp Ordner so eine Indexzeile in der Standardkonfiguration mitbringt, wird er im Normalfall auch gefunden werden. In der ersten MsgBox wird der aktuelle Wert ausgegeben, danach wird er mit einem anderen Wert „Ein Test“ belegt und erneut ausgegeben. Diese Änderung wird allerdings nicht gespeichert, so dass die Originaleinstellungen unverändert bleiben.

```
' Skript: GetSetObjAttribByName
' Autor: Matthias Thiele
' Erstellt: 05.07.2010
' Letzte Änderung: 05.07.2010
'
' Dieses Programm zeigt die Verwendung
' der Hilfsfunktionen zum Lesen oder
' Schreiben von Indexzeilen über den
' Gruppennamen


Option Explicit

Dim ELO
Set ELO = CreateObject("ELO.office")
Call ELO.PrepareObjectEx(2, 0, 0)
MsgBox GetObjAttribByName(ELO, "ELOINDEX")
Call SetObjAttribByName(ELO, "ELOINDEX", "Ein Test")
MsgBox GetObjAttribByName(ELO, "ELOINDEX")



' Schreibt den Wert "AttribValue" in die
' Indexzeile mit dem Gruppennamen "AttribName".
' Der erste Parameter enthält das ELO Objekt.
Sub SetObjAttribByName(ELO, AttribName, AttribValue)
  Dim i, LocalName

  LocalName = LCase(AttribName)
  For i = 0 to 49
```

```
    If LCase(ELO.GetObjAttribKey(i)) = LocalName Then
      Call ELO.SetObjAttrib(i, AttribValue)
      Exit For
    End If
  Next
End Sub

' Liest den aktuellen Wert der Indexzeile mit
' dem Namen "AttribName". Der erste Parameter
' enthält das ELO Objekt
Function GetObjAttribByName(ELO, AttribName)
  Dim i, LocalName

  GetObjAttribByName = ""
  LocalName = LCase(AttribName)
  For i = 0 to 49
    If LCase(ELO.GetObjAttribKey(i)) = LocalName Then
      GetObjAttribByName = ELO.GetObjAttrib(i)
      Exit For
    End If
  Next
End Function
```

Ein Fortschrittsbalken mit VB Script

Normalerweise sollte man darauf achten, dass ein Anwender nicht lange auf das Ergebnis einer Aktion warten muss. Manchmal gibt es aber lang laufende Vorgänge, die nicht weiter optimiert werden können. In diesem Fall ist es gut, wenn man dem Anwender zumindest eine Statusanzeige bieten kann, die ihn über den Fortschritt der Arbeiten unterrichtet. Das hat gleich zwei Vorteile: er wird nicht unsicher sein, ob das System noch arbeitet oder irgendwie „hängt“ und er kann entscheiden, ob er das Ende der Arbeiten abwartet oder in der Zwischenzeit eine andere Aktivität ausführt.

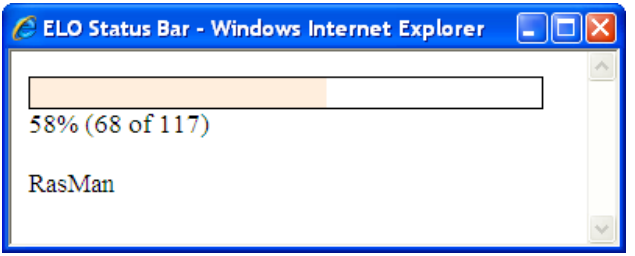
Hinweis: falls ein Skript eine umfangreichere Aktion ausführt, sollte man unbedingt darauf achten, dass der Anwender weiß, dass er während dieser Zeit nicht weiter im ELO arbeiten darf. Wenn er das doch tut und ungeduldig im Programm rumklickt, löst er Ereignisse in der OLE Schnittstelle aus, die zu erheblichen Fehlfunktionen im Skript führen können.

So eine Fortschrittsanzeige funktioniert natürlich nur, wenn Sie die Gesamtdauer abschätzen können. Eine Statusanzeige, die wild umherspringt und einmal eine Restdauer von 5 Minuten und direkt danach wieder eine Restdauer von 2 Tagen anzeigt, löst beim Anwender im besten Fall Belustigung aus. Wenn die Gesamtdauer unbekannt ist und auch nicht gut abgeschätzt werden kann, sollten Sie statt einer Fortschrittsanzeige lieber eine Aktivitätsanzeige verwenden.

Eine universell verwendbare Fortschrittsanzeige muss drei Funktionen zur Verfügung stellen:

1. Zum Start muss eine Anzeige mit 0% eingeblendet werden.
2. In regelmäßigen Abständen muss die Anzeige auf den aktuellen Bearbeitungsstand aktualisiert werden.
3. Nach dem Abschluss muss die Anzeige wieder entfernt werden.

Da die Möglichkeiten für ein Anwenderinterface unter VB-Script sehr begrenzt sind, muss man sich hier etwas anderes überlegen. Ein möglicher Ansatz wäre die Verwendung eines kleinen Visual Basic Programms, welches dann über die COM Schnittstelle (mittels CreateObject) angesprochen werden kann. Das hätte aber den Nachteil, dass Sie ein weiteres Programm installieren, registrieren und verteilen müssen. Es gibt aber auch einen Weg über den Internet Explorer. Er kann durch ein Skript ferngesteuert werden und er steht auf jeden Windows Rechner zur Verfügung.



Die Anzeige des Internet Explorers kann durch folgende Skriptbefehle erfolgen:

```
sub PrepareStatus(Titel, MaxValue, StartMessage)
  Set StatusExplorer =
WScript.CreateObject("InternetExplorer.Application")
  StatusExplorer.Navigate "about:blank"
  StatusExplorer.ToolBar = 0
  StatusExplorer.StatusBar = 0
  StatusExplorer.Width=370
  StatusExplorer.Height = 160
  StatusExplorer.Left = 0
  StatusExplorer.Top = 0

  Do While (StatusExplorer.Busy)
    Call ELO.Sleep(0, 200)
  Loop

  StatusExplorer.Visible = 1
  StatusExplorer.Document.Body.InnerHTML = StartMessage
  StatusExplorer.Document.Title = Titel

  StatusTotal = MaxValue
```

```
  StatusCount = 0
end sub
```

Diese Subroutine hat drei Parameter, den Fenstertitel, den Wert für die 100% Anzeige und eine Startnachricht, die angezeigt wird bevor die erste Fortschrittsanzeige erfolgt (diese kann auch leer sein). Der MaxValue Wert für die 100% Anzeige sollte auf die Anzahl der Schritte eingestellt werden, die im Folgenden aufgerufen werden. Wenn in einer Aktion 200 Dokumente bearbeitet werden sollen und der MaxValue Wert auf 200 eingestellt wird, dann führt ein StatusUpdate von 100 zu einer 50% Anzeige.

Hinweis: Das Explorerfenster wird an einer festen Position dargestellt. Zur Verbesserung könnte man noch die aktuelle Bildschirmauflösung ermitteln und es zentrieren. Wenn mehrere Monitore vorhanden sind, kann man aber nicht leicht entscheiden, auf welchem die Anzeige erfolgen soll.

```
Computer = "."
Set WMIService = GetObject("Winmgmts:\\" & Computer & "\root\cimv2")
Set Items = WMIService.ExecQuery("Select * From
Win32_DesktopMonitor")
For Each Item in Items
   Horizontal = Item.ScreenWidth
   Vertical = Item.ScreenHeight
   Msg = Msg & "Width: " & Horizontal & "  Heigth: " & Vertical & vbCrLf
Next

MsgBox Msg
```

Mittels CreateObject(“InternetExplorer.Application”) wird ein Browserfenster erzeugt und in der Variablen StatusExplorer gespeichert. In den folgenden Schritten wird die Anzeige und Größe des Browserfensters eingestellt. Anschließend wird in das Property InnerHTML die Startnachricht geschrieben.

Das Schließen des Browserfensters ist ebenfalls relativ einfach. Es kann optional noch eine Abschlussnachricht gesetzt und für eine einstellbare Dauer angezeigt werden. Danach wird das Browserfenster geschlossen.

```
Sub EndStatus(ReadyMessage, ShowDelay)
  StatusExplorer.Document.Body.InnerHTML = ReadyMessage
  Call ELO.Sleep(0, ShowDelay)
  StatusExplorer.Quit
end sub
```

Der Aufruf von EndStatus hat zwei Parameter, die Abschlussnachricht und die Anzeigedauer für diese Nachricht in Millisekunden bevor das Fenster geschlossen wird.

Die eigentliche Fortschrittsanzeige muss den Prozentwert errechnen, geeigneten HTLM Code für den Fortschrittsbalken erzeugen und an den Internet Explorer übergeben. Den HTLM Code werden wir hier nicht weiter untersuchen, er besitzt ein DIV Tag mit einstellbarer Länge für den Fortschrittsbalken. Diese Länge wird über die Prozentzahl gesteuert.

```
sub UpdateStatus(NewStatusCount, Message)
   Dim Prozent
   StatusCount = NewStatusCount
   if StatusCount > StatusTotal then StatusCount = StatusTotal
   Prozent = CLng((StatusCount / StatusTotal) * 100)
   StatusExplorer.Document.Body.InnerHTML =_
     "<div style=""border:solid 1px;width:302px""><div style=""width:"_
     & 3 * prozent & "px;background-color:#ffeedd""></div></div><div>"_
     & Prozent & "% (" & StatusCount & " of " & StatusTotal _
     & ")</div><div> </div><div>" & Message & "</div>"
end Sub
```

Der Aufruf besitzt zwei Parameter, einmal den neuen Statuswert zur Ermittlung des Prozentwerts und einen weiteren zur Anzeige des aktuell bearbeiteten Eintrags (z.B. der aktuelle Dokumentenname).

Es gibt noch eine vierte Funktion, als Komfortfunktion: wenn die Anzahl der Aufrufe genau bekannt ist, muss man die aktuelle Position nicht selber

speichern sondern kann einfach per IncrementStatus Schrittweise weiterlaufen.

```
sub IncrementStatus(Message)
   call UpdateStatus(StatusCount + 1, Message)
end sub
```

Jetzt fehlt nur noch das Rahmenprogramm, welches diese Funktionen verwendet. Hier wird ein kleines Beispiel genutzt, welches vom Betriebssystem eine Liste aller Systemdienste anfordert und diese dann abarbeitet. Die Bearbeitung beschränkt sich dabei aber auf eine künstliche Wartezeit von 100 Millisekunden.

```
 Dim Computer, Cnt, ColServices, ObjService
```

Hier wird die Liste der Systemdienste angefordert

```
 Computer = "."
 Set ColServices = GetObject("winmgmts:\\" & Computer_
  & "\root\cimv2"). _
  ExecQuery("Select * from Win32_Service")
```

Anschließend wird erst einmal durchgezählt, wie viele es gesamt sind. Hierdurch wird der MaxValue Wert für die Statusanzeige ermittelt.

```
 Cnt = 0
 For Each objService in colServices
  Cnt = Cnt + 1
 Next
```

Nun wird die Statusanzeige gestartet.

```
 call PrepareStatus("ELO Status Bar", Cnt,_
         "Überprüfung aller Betriebssystemdienste. " & _
         vbcrlf & "Dieser Vorgang kann einige Minuten dauern.")
```

```
  call ELO.Sleep(0, 200)
  For Each ObjService in ColServices
    call ELO.Sleep(0, 100)
```

Hier wird für jeden Service eine Aktualisierung der Statusanzeige ausgelöst.
Der Servicename wird unterhalb des Statusbalkens eingeblendet

```
    call IncrementStatus(ObjService.Name)
  Next
```

Am Ende wird das Statusfenster wieder geschlossen

```
  call EndStatus("Liste vollständig abgearbeitet.", 1000)
```

Das vollständige Skript sieht dann so aus:

```
' Skript: DemoStatusbar
' Autor: Matthias Thiele
' Erstellt: 11.06.2010
' Letzte Änderung: 13.06.2010
'
' Dieses Programm demonstriert die Verwendung eines
Fortschrittsbalkens,
' realisiert mit dem Internet Explorer.

option explicit

Dim ELO
set ELO = CreateObject("ELO.office")
call Main


' In der Hauptroutine wird aus dem Betriebssystem zuerst eine Liste
' aller Systemdienste eingelesen. Diese Liste wird dann unter
' Verwendung der Statusanzeige "abgearbeitet", wobei die Bearbeitung
' durch eine künstliche Wartezeit (Sleep) von 100 Millisekunden
' simuliert wird.
```

```
sub Main
  Dim Computer, Cnt, ColServices, ObjService
  Computer = "."
  Set ColServices = GetObject("winmgmts:\\" & Computer_
    & "\root\cimv2"). _
    ExecQuery("Select * from Win32_Service")

  Cnt = 0
  For Each objService in colServices
    Cnt = Cnt + 1
  Next

  call PrepareStatus("ELO Status Bar", Cnt,_
          "Überprüfung aller Betriebssystemdienste. " & _
          vbcrlf & "Dieser Vorgang kann einige Minuten dauern.")

  call ELO.Sleep(0, 200)
  For Each ObjService in ColServices
    call ELO.Sleep(0, 100)
    call IncrementStatus(ObjService.Name)
  Next

  call EndStatus("Liste vollständig abgearbeitet.", 1000)
end sub

' Hier beginnt der Funktionsbereich für die Statusanzeige
' Die Statusanzeige wird folgender Reihenfolge aufgerufen
'   PrepareStatus
'   mehrfaches IncrementStatus
'   EndStatus
'
' Globale Variablen zur Speicherung des aktuellen Status
dim StatusExplorer, StatusTotal, StatusCount

' Zeigt die Statusanzeige an und speichert den 100% Wert
sub PrepareStatus(Titel, MaxValue, StartMessage)
  Set StatusExplorer = CreateObject("InternetExplorer.Application")
  StatusExplorer.Navigate "about:blank"
  StatusExplorer.ToolBar = 0
  StatusExplorer.StatusBar = 0
  StatusExplorer.Width=400
  StatusExplorer.Height = 160
  StatusExplorer.Left = 0
  StatusExplorer.Top = 0
```

```
  Do While (StatusExplorer.Busy)
    call ELO.Sleep(0, 200)
  Loop

  StatusExplorer.Visible = 1
  StatusExplorer.Document.Body.InnerHTML = StartMessage
  StatusExplorer.Document.Title = Titel

  StatusTotal = MaxValue
  StatusCount = 0
end sub


' Verlängert die Statusanzeige um einen Schritt
sub IncrementStatus(Message)
    call UpdateStatus(StatusCount + 1, Message)
end sub

' Aktualisiert die Länge der Statusanzeige auf eine neuen Wert
sub UpdateStatus(NewStatusCount, Message)
    Dim Prozent
    StatusCount = NewStatusCount
    if StatusCount > StatusTotal then StatusCount = StatusTotal
    Prozent = CLng((StatusCount / StatusTotal) * 100)
    StatusExplorer.Document.Body.InnerHTML =_
      "<div style=""border:solid 1px;width:302px""><div style=""width:"_
      & 3 * prozent & "px;background-color:#ffeedd""></div></div><div>"_
      & Prozent & "% (" & StatusCount & " of " & StatusTotal _
      & ")</div><div> </div><div>" & Message & "</div>"
end Sub


' Schließt die Statusanzeige wieder
Sub EndStatus(ReadyMessage, ShowDelay)
  StatusExplorer.Document.Body.InnerHTML = ReadyMessage
  call ELO.Sleep(0, ShowDelay)
  StatusExplorer.Quit
end sub
```

Objektorientierte Programmierung mit VB-Script

Das Beispiel zur Anzeige eines Statusbalkens mit dem Internet Explorer zeigt erste Grenzen der einfachen Skriptprogrammierung auf. Es werden zwar leicht verständliche Funktionen zur Verwendung angeboten. Aber es gibt auch eine Reihe globaler Variablen zur Speicherung des internen Status der Anzeige. Bei dieser Art der Skriptprogrammierung kann nicht verhindert werden, dass andere Programmteile irrtümlich oder absichtlich auf die globalen Variablen zugreifen, diese verändern und so für Störungen sorgen. Weiterhin enthält die Schnittstelle Funktionsnamen, die künstlich aus einer globalen Funktionsbeschreibung und einer lokalen Funktion zusammengesetzt sind: „Update“-„Status“. Wenn man keine Klassen hat, dann ist das eine allgemein verwendete Technik um einen gewissen logischen Zusammenhalt der Funktionen zu erreichen. Wenn der Name nur Update lauten würde, könnte man nicht leicht feststellen, was hier überhaupt aktualisiert wird. Zudem bestünde ein großes Risiko, dass der Funktionsname mit anderen Update Funktionen, z.B. Update(User), Update(Document) etc. kollidieren würde.

An dieser Stelle sollte man über Objektorientierte Programmierung und Stichworte wie Kapselung nachdenken. VB-Script ist nicht die beste Sprache für die Demonstration von OOP Techniken, aber einige praktische Fähigkeiten stehen zur Verfügung und sollten genutzt werden wenn es sinnvoll ist.

Leider fehlt genau das wichtigste Merkmal: die Vererbung. Deshalb wird VB-Script von vielen nicht als Objektorientierte Sprache angesehen. Dieses Problem führt dazu, dass in den Beispielen an ein paar Stellen eine Mischung aus Prozeduraler Programmierung und OOP Techniken zur Anwendung kommt.

Mini OOP Skript in VB

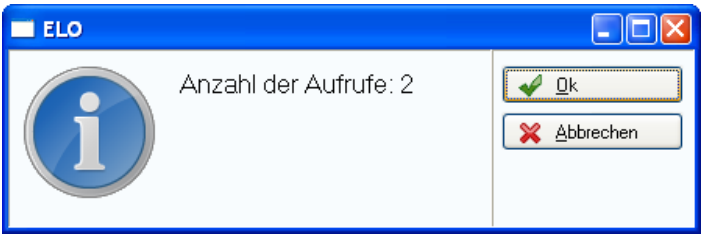
Der Aufbau einer eigenen Klasse in VB-Script ist sehr einfach. Es gibt einen Rahmen, in dem alle Variablen, Funktionen und Subroutinen der Klasse aufgeführt werden.

```
Class MeineErsteVbsKlasse
   ' hier folgen die klasseninternen Deklarationen
End Class
```

Die klasseninternen Deklarationen können privat oder öffentlich sein. Private Variablen und Funktionen sind nur innerhalb der Klasse sichtbar und verwendbar, öffentliche Funktionen können auch von außen verwendet werden.

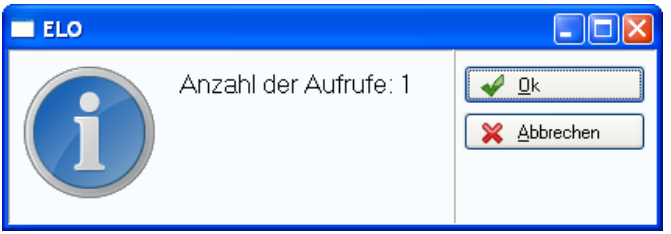
Ein einfaches Beispiel zur Berechnung irgendwelcher Daten ohne Klassen könnte so aussehen:

```
option explicit

Dim ELO, Message, UsageCount
Set ELO = CreateObject("ELO.office")


call ProcessData
call ProcessData
Message = "Anzahl der Aufrufe: " & UsageCount
call ELO.MsgBox(Message, "ELO", vbOk)


Sub ProcessData
  ' Führt umfangreiche Berechnungen aus
  UsageCount = UsageCount + 1
End Sub
```



Die Subroutine ProcessData soll eine komplexe Bearbeitung durchführen. Zu Abrechnungszwecken soll zusätzlich ein Zähler mitgeführt werden, der protokolliert, wie oft diese Berechnung ausgeführt wurde (UsageCount). Problematisch dabei ist, dass dieser Zähler völlig ungeschützt im Programm Code liegt und von beliebiger Stelle aus absichtlich oder irrtümlich verändert werden kann.

```
option explicit

Dim ELO, Message, UsageCount
Set ELO = CreateObject("ELO.office")


call ProcessData

' Böse Aktion, kann aber nicht verhindert werden:
UsageCount = 0

call ProcessData

Message = "Anzahl der Aufrufe: " & UsageCount
call ELO.MsgBox(Message, "ELO", vbOk)


Sub ProcessData
  ' Führt umfangreiche Berechnungen aus
  UsageCount = UsageCount + 1
End Sub
```



Durch die Verwendung einer Klasse kann man den Zugriff auf diese Variable schützen.

```
option explicit

Dim ELO, Message, Test
Set ELO = CreateObject("ELO.office")

Set Test = New TestClass

call Test.ProcessData
call Test.ProcessData
Message = "Anzahl der Aufrufe: " & Test.GetUsageCount
call ELO.MsgBox(Message, "ELO", vbOk)


Class TestClass

  ' Eine private Variable, auf sie kann von außen
  ' nicht zugegriffen werden
  Private UsageCount

  ' Eine öffentliche Methode
  Public Sub ProcessData
    ' Führt umfangreiche Berechnungen aus
    UsageCount = UsageCount + 1
  End Sub

  ' Eine öffentliche Funktion
  Public Function GetUsageCount
    GetUsageCount = UsageCount
  End Function

End Class
```

In der Klasse TestClass gibt es eine private Variable UsageCount. Sie ist von außen nicht sichtbar und kann deshalb extern auch nicht verändert werden. Die öffentliche Klassenfunktion ProcessData erhöht den Zähler bei jeder Ausführung. Damit diese Variable von außen abgefragt werden kann, gibt es zusätzlich noch eine öffentliche Zugriffsfunktion GetUsageCount.

Ein so kleines Beispiel kann kaum die Vorteile der Objektorientierten Programmierung zeigen. Zudem enthält dieses Beispiel auch nur einen ganz kleinen Ausschnitt aus der OOP Welt. In einer Skript-Umgebung werden Sie im Normalfall auch keine komplexen Klassenhierarchien aufbauen. Aber auch einfache Skripte können zumindest von der Kapselung profitieren.

Der Fortschrittsbalken in einer OOP Variante

Das DemoStatusBar Skript kann in der OOP Version vor allem durch den Schutz für die internen Variablen und durch eine bessere Benennung der Funktionen profitieren. Bei den Einzelfunktionen wurden künstlich zusammengesetzte Funktionsnamen wie z.B. PrepareStatus verwendet. Ein einfaches Show an dieser Stelle wäre zum einen für den Leser nicht verständlich (Was wird denn angezeigt?) und hätte zum anderen ein hohes Kollisionsrisiko mit gleichartigen Funktionen aus anderen Bereichen. In der OOP Version kann man ohne Probleme Show als Name verwenden, weil dieser Funktionsaufruf immer zusammen mit dem Objekt steht:

```
Set Status = new ExplorerStatusBar
call Status.Show(...)

For Each ObjService in ColServices
  call Status.Increment(…)
Next

call Status.Hide(…)
```

Hier nun das DemoStatusBarOop Skript komplett:

```
' Skript: DemoStatusbarOop
```

```
' Autor: Matthias Thiele
' Erstellt: 12.06.2010
' Letzte Änderung: 15.06.2010
'
' Dieses Programm demonstriert die Verwendung eines
Fortschrittsbalkens,
' realisiert mit dem Internet Explorer.


option explicit

Dim ELO
set ELO = CreateObject("ELO.office")
call Main


' In der Hauptroutine wird aus dem Betriebssystem zuerst eine Liste
' aller Systemdienste eingelesen. Diese Liste wird dann unter
' Verwendung der Statusanzeige "abgearbeitet", wobei die Bearbeitung
' durch eine künstliche Wartezeit (Sleep) von 100 Millisekunden
' simuliert wird.

sub Main
  Dim Cnt, ColServices, ObjService, Status
  Set ColServices = GetObject("winmgmts:\\.\root\cimv2"). _
    ExecQuery("Select * from Win32_Service")

  Cnt = 0
  For Each objService in colServices
    Cnt = Cnt + 1
  Next

  Set Status = new ExplorerStatusBar
  call Status.Show("ELO Status Bar", Cnt,_
            "Überprüfung aller Betriebssystemdienste. " & _
            vbcrlf & "Dieser Vorgang kann einige Minuten dauern.")

  call ELO.Sleep(0, 200)
  For Each ObjService in ColServices
    call ELO.Sleep(0, 100)
    call Status.Increment(ObjService.Name)
  Next

  call Status.Hide("Liste vollständig abgearbeitet.", 1000)
end sub
```

```
' Hier beginnt der Funktionsbereich für die Klasse der Statusanzeige
' Die Statusanzeige wird durch Methodenaufrufe in folgender
' Reihenfolge  verwendet
'
'   Show
'   mehrfaches Increment oder Update
'   Hide

Class ExplorerStatusBar

Private Explorer, Total, Count

' Zeigt die Statusanzeige an und speichert den 100% Wert
Public Sub Show(Titel, MaxValue, StartMessage)
  Set Explorer = CreateObject("InternetExplorer.Application")
  Explorer.Navigate "about:blank"
  Explorer.ToolBar = 0
  Explorer.StatusBar = 0
  Explorer.Width=370
  Explorer.Height = 160
  Explorer.Left = 0
  Explorer.Top = 0

  Do While (Explorer.Busy)
    call ELO.Sleep(0, 200)
  Loop

  Explorer.Visible = 1
  Explorer.Document.Body.InnerHTML = StartMessage
  Explorer.Document.Title = Titel

  Total = MaxValue
  Count = 0
End Sub


' Verlängert die Statusanzeige um einen Schritt
Public Sub Increment(Message)
    call Update(Count + 1, Message)
End Sub
```

```
' Aktualisiert die Länge der Statusanzeige auf eine neuen Wert
Public Sub Update(NewStatusCount, Message)
   Dim Prozent
   Count = NewStatusCount
   if Count > Total then Count = Total
   Prozent = CLng((Count / Total) * 100)
   Explorer.Document.Body.InnerHTML = _
    "<div style=""border:solid 1px;width:302px""><div style=""width:" _
    & 3 * prozent & "px;background-color:#ffeedd""></div></div><div>"_
    & Prozent & "% (" & Count & " of " & Total _
    & ")</div><div> </div><div>" & Message & "</div>"
End Sub


' Schließt die Statusanzeige wieder
Public Sub Hide(ReadyMessage, ShowDelay)
  Explorer.Document.Body.InnerHTML = ReadyMessage
  call ELO.Sleep(0, ShowDelay)
  Explorer.Quit
End Sub

End Class
```

Über die Anwendung Objektorientierter Methoden in einfache Skripten kann man sich streiten. Bei einfachen kleinen Skripten, die nur einen begrenzten Funktionsumfang besitzen, ist das in manchen Fällen vielleicht unnötiger Ballast (obwohl der Zusatzaufwand ja nur minimal ist). Sobald Sie aber Funktionen erstellen, die möglicherweise in anderen Skripten wiederverwendet werden sollen, ist die Kapselung ein wertvoller Weg um unerwartete Kollisionen und Nebenwirkungen zu vermeiden.

Warteanzeige im Internet Explorer

Wenn man die genaue Anzahl der Schritte nicht kennt und auch nicht abschätzen kann, dann ist ein Fortschrittsbalken eine schlechte Form der Rückmeldung. In diesem Fall sollte man besser eine Anzeige einsetzen, die zwar eine Aktivität signalisiert aber keine (falschen) Hinweise zur Restdauer gibt. Ein gängiger Weg hierfür ist ein hin- und herlaufender Balken. Das folgende Skript ist eine leichte Abwandlung des Fortschrittsbalkens. Lediglich

die Methode Increment fehlt und die Methode Update hat nur noch einen Status-Text ohne Position.

```
' Skript: DemoWaitSign
' Autor: Matthias Thiele
' Erstellt: 24.06.2010
' Letzte Änderung: 25.06.2010
'
' Dieses Programm zeigt einen Wartedialog
' im Internet Explorer an


option explicit

Dim ELO
set ELO = CreateObject("ELO.office")
call Main


' In der Hauptroutine wird aus dem Betriebssystem zuerst eine Liste
' aller Systemdienste eingelesen. Diese Liste wird dann unter
' Verwendung der Statusanzeige "abgearbeitet", wobei die Bearbeitung
' durch eine künstliche Wartezeit (Sleep) von 100 Millisekunden
' simuliert wird.

sub Main
  Dim ColServices, ObjService, Status
  Set ColServices = GetObject("winmgmts:\\.\root\cimv2"). _
    ExecQuery("Select * from Win32_Service")

  Set Status = new ExplorerWaitBar
  call Status.Show("ELO Wait Bar",_
            "Überprüfung aller Betriebssystemdienste. " & _
            vbcrlf & "Dieser Vorgang kann einige Minuten dauern.")

  call ELO.Sleep(0, 200)
  For Each ObjService in ColServices
    call ELO.Sleep(0, 100)
    call Status.Update(ObjService.Name)
  Next

  call Status.Hide("Liste vollständig abgearbeitet.", 1000)
end sub
```

```
' Hier beginnt der Funktionsbereich für die Klasse der Statusanzeige
' Die Statusanzeige wird durch Methodenaufrufe in folgender
' Reihenfolge verwendet
'
'   Show
'   mehrfaches Update
'   Hide

Class ExplorerWaitBar

Private Explorer

' Zeigt die Statusanzeige an
Public Sub Show(Titel, StartMessage)
  Set Explorer = CreateObject("InternetExplorer.Application")
  Explorer.Navigate "about:blank"
  Explorer.ToolBar = 0
  Explorer.StatusBar = 0
  Explorer.Width=370
  Explorer.Height = 160
  Explorer.Left = 0
  Explorer.Top = 0

  Do While (Explorer.Busy)
    call ELO.Sleep(0, 200)
  Loop

  Explorer.Visible = 1
  call PrepareHtml(Titel, StartMessage)
End Sub



' Aktualisiert die Warteanzeige
Public Sub Update(Message)
  Dim Doc, Div

  Set Doc = Explorer.Document
  Set Div = Doc.getElementById("msg")
  Div.innerHTML = Message
End Sub
```

```
' Schließt die Statusanzeige wieder
Public Sub Hide(ReadyMessage, ShowDelay)
  Explorer.Document.Body.InnerHTML = ReadyMessage
  call ELO.Sleep(0, ShowDelay)
  Explorer.Quit
End Sub

Private Sub PrepareHtml(Title, StartMessage)
  Dim Hdr, Body, Start, Doc

  Hdr ="<head><title>" & Title & "</title>" & _
      "<script type=""text/javascript"">" & _
      "var leftpos = 0;" & _
      "var innerpos = 0;" & _
      "var direction = 1;" & _
      "function tick() {" & _
      " var outer = document.getElementById(""outer"");" & _
      " var inner = document.getElementById(""inner"");" & _
      " if (inner && outer) {" & _
      " if (direction == 1) {" & _
      "   if (innerpos < 72) {" & _
      "     innerpos++;" & _
      "     inner.style.left = innerpos + ""px"";" & _
      "   } else {" & _
      "     leftpos++;" & _
      "     outer.style.left = leftpos + ""px"";" & _
      "     if (leftpos > 120) {" & _
      "       direction = -1;" & _
      "     }" & _
      "   }" & _
      " } else {" & _
      "   if (innerpos > 0) {" & _
      "     innerpos--;" & _
      "     inner.style.left = innerpos + ""px"";" & _
      "   } else {" & _
      "     leftpos--;" & _
      "     outer.style.left = leftpos + ""px"";" & _
      "     if (leftpos < 1) {" & _
      "       direction = 1;" & _
      "     }" & _
      "   }" & _
      " }" & _
      "}" & _
      "}" & _
      "</script></head>"
```

```
  Body = "<body scroll=""no"" onload=""setInterval('tick()', 20);"">" & _
       "<div style=""border:solid 1px;width:203px;color:#808080"">" & _
       "<div id=""outer"" style=""position:relative;left:0px;" & _
       "width:80px;background-color:#f0f0fa;height:30px"">" & _
       "<div id=""inner"" style=""position:relative;left:0px;width:8px;" & _
       "background-color:#00ff80;height:30px"">" & _
       "<div style=""position:relative;left:0px;width:4px;" & _
       "background-color:#ffc840;height:30px"">" & _
       " </div></div></div></div><div id=""msg"">" & _
       StartMessage & "</div></body>"

  Start = "<html>" & Hdr & Body & "</html>"
  Set Doc = Explorer.Document
  Doc.Open
  Call Doc.Write(Start)
  Doc.Close
End Sub

End Class
```

Fehlersuche in Skripten

Langsam werden die Skriptbeispiele aufwendiger, deshalb ist es nun an der Zeit, sich ein paar Gedanken über die Fehlersuche bei der Skriptentwicklung zu machen. Leider gibt es im VB-Script Umfeld keinen richtigen Debugger, wie man ihn von modernen Entwicklungsumgebungen her kennt. Aber ELOoffice stellt Ihnen ein paar Hilfsmittel zur Verfügung, die das Leben hier etwas einfacher machen.

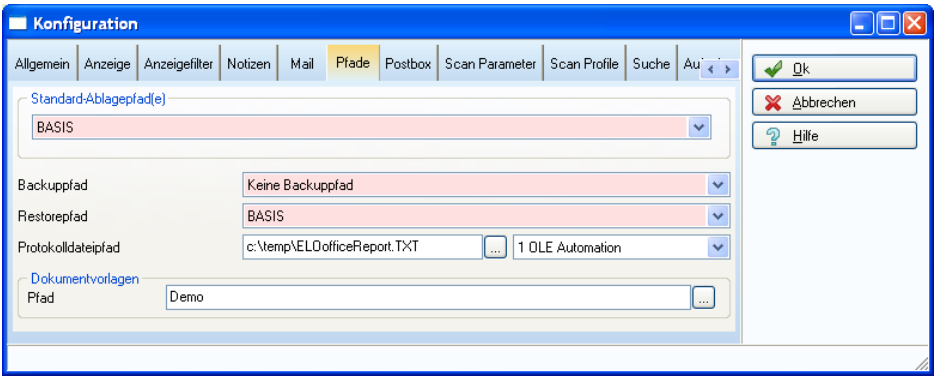
Debugging mit Message Boxen

Eine naheliegende Form der Fehlersuche liegt darin, dass Sie an kritischen Stellen MsgBox Einträge in Ihren Code aufnehmen, der alle wichtigen Parameter ausgibt. Für eine kurze einfache Kontrolle ist das ein schneller Weg um an Informationen zu kommen. Der Nachteil liegt darin, dass der normale Programmablauf erheblich verzögert wird. Wenn eine Vielzahl von Mitteilungen kommt, wird die Fehlersuche extrem zeitraubend. Zudem ist eine einmal mit „Ok“ bestätigte Message Box weg und mit ihr der Meldungstext. Wenn Sie sich ihn nicht merken konnten und nicht aufgeschrieben haben, müssen Sie wieder von vorne anfangen.

ELOoffice Client Report

Wenn Sie schon mal unseren Support zu einem Problem angerufen haben, dann ist Ihnen mit hoher Wahrscheinlichkeit auch der Client Report bekannt. In der Konfiguration unter „Pfade“ befindet sich eine Eingabezeile für den „Protokolldateipfad“. Hier tragen Sie den Dateinamen für die Reportdatei mit voller Pfadinformation ein, z.B. C:\Temp\ELOofficeReport.txt. Dahinter finden Sie eine Auswahlliste, die im Anfangszustand auf „0 Kein Report“ eingestellt ist. Der Support fordert im Normalfall einen Report mit „512 Voller Report“ an, dieser enthält aber viele interne Daten, die für die Skriptentwicklung nicht relevant sind. Wählen Sie am besten „1 OLE Automation“. In dieser Betriebsart werden alle OLE Automation Aufrufe

protokolliert, in vielen Fällen mit Parametern und Rückgabewerten (leider aber nicht immer).



Eine Suche mit dem weiter oben aufgeführten Skript zur Verschlagwortungsvorbelegung löst dann folgende Reportausgaben aus:

```
2010-06-10 13:30:43.460 :  0      CallScript : 13 : EVT_Vorbelegung
2010-06-10 13:30:43.480 :  0      Enter:GetActionKey
2010-06-10 13:30:43.480 :  0      Exit: GetActionKey Result=32779
2010-06-10 13:30:43.480 :  0      Script done: 13
2010-06-10 13:30:43.590 :  0      CallScript : 13 : EVT_Vorbelegung
2010-06-10 13:30:43.590 :  0      Enter:GetActionKey
2010-06-10 13:30:43.590 :  0      Exit: GetActionKey Result=32789
2010-06-10 13:30:43.590 :  0      Script done: 13
2010-06-10 13:34:42.890 :  0      CallScript : 13 : EVT_Vorbelegung
2010-06-10 13:34:42.910 :  0      Enter:GetActionKey
2010-06-10 13:34:42.910 :  0      Exit: GetActionKey Result=24
2010-06-10 13:34:42.910 :  0      Script done: 13
2010-06-10 13:34:43.000 :  0      CallScript : 13 : EVT_Vorbelegung
2010-06-10 13:34:43.020 :  0      Enter:GetActionKey
2010-06-10 13:34:43.020 :  0      Exit: GetActionKey Result=32788
2010-06-10 13:34:43.020 :  0      Enter:GetObjMaskNo
2010-06-10 13:34:43.020 :  0      Exit: GetObjMaskNo Result=8
2010-06-10 13:34:43.020 :  0      Enter:GetObjAttrib AttribNo=4
2010-06-10 13:34:43.020 :  0      Exit: GetObjAttrib Result=
2010-06-10 13:34:43.020 :  0      Enter:SetObjAttrib AttribNo=2
AttribText=Contelo GmbH
2010-06-10 13:34:43.020 :  0      Exit: SetObjAttrib Result=1
2010-06-10 13:34:43.020 :  0      Script done: 13
2010-06-10 13:34:43.100 :  0      CallScript : 13 : EVT_Vorbelegung
```

```
2010-06-10 13:34:43.100 :  0      Enter:GetActionKey
2010-06-10 13:34:43.100 :  0      Exit: GetActionKey Result=32778
2010-06-10 13:34:43.110 :  0      Script done: 13
2010-06-10 13:34:45.150 :  0      CallScript : 13 : EVT_Vorbelegung
2010-06-10 13:34:45.150 :  0      Enter:GetActionKey
2010-06-10 13:34:45.150 :  0      Exit: GetActionKey Result=32779
2010-06-10 13:34:45.170 :  0      Script done: 13
2010-06-10 13:34:45.280 :  0      CallScript : 13 : EVT_Vorbelegung
2010-06-10 13:34:45.290 :  0      Enter:GetActionKey
2010-06-10 13:34:45.290 :  0      Exit: GetActionKey Result=32789
2010-06-10 13:34:45.290 :  0      Script done: 13
```

Man kann im Report sehr gut sehen, welche Befehle aufgerufen worden und welche Daten geschrieben wurden, z.B. um 13:34:43.020 wurde mit SetObjAttrib in die Indexzeile 2 der Wert „Contelo GmbH“ geschrieben.

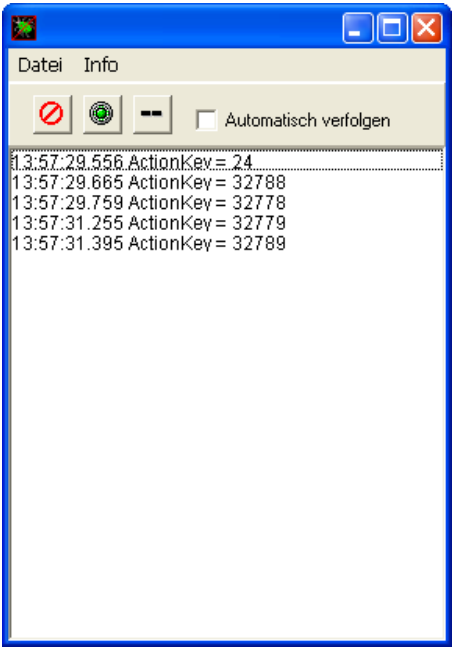
Ein ganz wichtiger Vorteil des Reports liegt auch darin, dass Sie nachträglich auch Werte kontrollieren können, an die Sie beim Start nicht gedacht hatten. Bei dem MessageBox Ansatz hätten Sie diese also auch nicht ausgegeben oder nicht notiert. Auch der zeitliche Ablauf bleibt weitestgehend erhalten, da die zusätzliche Protokollierung kaum zusätzliche Zeit benötigt. Der Report eignet sich damit auch hervorragend für die Untersuchung von Performance Problemen.

Im Normalfall finden Sie in dem Report nur die aufgerufenen OLE Automation Befehle. Falls Sie zusätzliche Informationen über interne Skriptzustände benötigen, können Sie den Befehl ELO.DebugOut verwenden um zusätzliche Texte in den Report aufzunehmen. Dieser Befehl hat zudem noch eine weitere sehr praktische Funktion, die Sie im nächsten Kapitel kennen lernen.

Ein Nachteil hat der Report – die Ergebnisse sind nicht sofort sichtbar. Man muss erst die Reportdatei öffnen und an die richtige Stelle gehen. Wenn man Schrittweise durch ein Programm läuft und jeweils den aktuellen Status prüfen möchte, kann das mühsam sein.

Einsatz des ELODebugOut Tools

Die dritte Variante ist die komfortabelste Möglichkeit um Ereignisse direkt anzeigen zu lassen. Im Download Bereich und im Tools Verzeichnis Ihrer Installations-DVD finden Sie ein Programm ELODebugOut.EXE. Wenn Sie dieses starten, dann wird jeder Aufruf von DebugOut im Skript direkt in einem Fenster angezeigt. Sie können in diesem Fall den zeitlichen Ablauf der einzelnen Aktionen leicht erkennen und müssen nicht jedes Mal neu eine Datei öffnen.



Ein Nachteil ist, dass diese Ausgabe nur Einträge enthält, die ausdrücklich per DebugOut Befehl gesendet wurden. In der Praxis ist bei komplizierten Problemen eine Kombination aus der Reportdatei und dem ELODebugOut Tool die beste Lösung.

Neues Dokument aus Vorlage erstellen

Wenn eine neues Word Dokument erstellt und abgelegt werden soll, z.B. eine Rechnung, dann hat der Anwender verschiedene Möglichkeiten. Zum einen kann er direkt Word öffnen, dort die Rechnungsvorlage aussuchen und öffnen, die Rechnung mit den benötigten Daten füllen und über die ELO Word Makros im Archiv ablegen.

Er kann aber auch im ELOoffice an die gewünschte Archivposition gehen, dort auf „Neues Dokument“ klicken und die Rechnungsvorlage aussuchen. Es öffnet sich dann automatisch Word – hier kann er die Daten einfüllen und speichern und schließen. Danach muss man im ELOoffice nur noch ein CheckIn ausführen.

Beide Varianten haben ihre Vor- und Nachteile, gemeinsam haben sie aber den Aufwand, dass eine spezielle Vorlage ausgesucht werden muss. Wenn an einem Arbeitsplatz immer ein bestimmter Dokumententyp erzeugt wird, dann kann man diese Vorauswahl auch per Skript erledigen.

Dieses Skript soll folgende Aktionen ausführen:

- Die Verschlagwortung für das neue Rechnungsdokument abfragen. Dabei soll die Rechnungsmaske bereits voreingestellt sein
- Ein ELO Dokumenteneintrag aus der eingegebenen Verschlagwortung im aktuell eingestellten Ordner erzeugen.
- Das Vorlagendokument im CheckOut Bereich anlegen und Word aufrufen.

Der Anwender muss nun nur noch den Zielordner auswählen, die Schaltfläche „Neue Rechnung“ anklicken, das Word Dokument ausfüllen und im ELO auf CheckIn klicken.

Als Initialisierung benötigt das Skript die Nummer der Rechnungsmaske, die interne ELO ObjektId des Vorlagendokuments und den Dokumententyp „Word“. Diese Informationen werden gemeinsam als Konstanten am Anfang

des Skripts deklariert, so dass sie später bei Bedarf leicht angepasst werden können.

```
const TemplateDoc = 31
const WordTypeId = 255
const InvoiceMaskNo = 8
```

Zur Ausführung muss das Skript zuerst prüfen, ob sich der Anwender in der Archivansicht befindet und ob ein Ordner ausgewählt ist. Die Ansicht wird über den Befehl „SelectView(0)“ ausgelesen, die Archivansicht hat den Wert 1. Der aktuell selektierte Eintrag kann durch „GetEntryId(-1)“ geprüft werden. Wenn der Wert kleiner als Null ist, dann ist kein Eintrag ausgewählt und es wird eine Fehlernachricht ausgegeben. Andernfalls wird durch die Funktion CreateEntry ein neues Dokument erzeugt.

```
if Elo.SelectView(0) = 1 then
  ParentId = Elo.GetEntryId(-1)
  if ParentId < 1 then
    call Elo.MsgBox("Bitte wählen Sie zuerst einen Zielordner aus", _
                "ELO", vbOkOnly or vbInformation)
  else
    call CreateEntry
  end if
else
  call Elo.MsgBox(_
  "Diese Funktion kann nur in der Archivansicht verwendet werden", _
  "ELO", vbOkOnly or vbInformation)
end if
```

Die Funktion CreateEntry erzeugt zuerst eine Verschlagwortung mit dem Dokumententyp „Word“ und der Verschlagwortungsmaske „Rechnung“. Diese wird dem Anwender zur Bearbeitung angezeigt (DoEditObject) und anschließend gespeichert. Die interne ELO Objektnummer des neuen Eintrags wird über „GetEntryId(-2) ermittelt. Danach wird dieser Eintrag gesperrt (CheckOut) und über „CopyExecTemplateFile“ Word aufgerufen.

```
Sub CreateEntry
  call Elo.PrepareObjectEx(0, WordTypeId, InvoiceMaskNo )
```

```
  Elo.ObjIndex = "#" & ParentId
  Elo.ObjShort = "Neue Rechnung"
  if Elo.DoEditObject() = 1 then
    if Elo.UpdateObject() < 0 then
      call Elo.MsgBox(_
           "Das neue Dokument konnte nicht angelegt werden", _
           "ELO", vbOkOnly or vbInformation)
    else
      Id = Elo.GetEntryId(-2)
      call Elo.LockObject(Id, 1)
      call Elo.GotoId(Id)
      CopyExecTemplateFile(Id)
    end if
  end if
End Sub
```

Zum Aufruf von Word muss die Datei der Dokumentenvorlage in das CheckOut Verzeichnis kopiert werden und anschließend zur Bearbeitung aktiviert werden. Dabei kann man aber nicht einfach den vorhandenen Dateinamen der Vorlage verwenden, da ELO im CheckOut Bereich einen speziell formatierten Dateinamen erwartet. Dieser setzt sich aus dem Archivnamen, der logischen Objektnummer und einem freien Namensanteil zusammen.

```
sub CopyExecTemplateFile(Id)
  Dim Source, Dest, fso, file, ArchiveName

  ArchiveName = Replace(Elo.GetArcName(), "_", " ")
  Source = Elo.GetDocumentPath(TemplateDoc, 0)
  Dest = Elo.GetPostDir() & "CheckOut\Rechnung_" _
          & ArchiveName & "_" & Id & Elo.SplitFileName(Source, 3)
  Set fso = CreateObject("Scripting.FileSystemObject")
  call fso.CopyFile(Source, Dest)
  Set file = fso.GetFile(Dest)
  file.Attributes = 0
  call Elo.DoExecuteEx(Dest, "", "", "Edit", -1)
end sub
```

Das komplette Skript sieht dann so aus:

```
' Skript: Neue Rechnung
```

```
' Autor: Matthias Thiele
' Erstellt: 14.06.2010
' Letzte Änderung: 14.06.2010
'
' Dieses Programm demonstriert die Erzeugung
' eines neuen Dokuments aus einer Vorlage

Option explicit

dim Elo, ArchiveName, Id
dim ParentId

' TemplateDoc enthält die logische ELO ObjektId der Vorlagendatei
' WordTypeId enthält den Dokumententyp (254...282, je nach Extension
setzen)
const TemplateDoc = 31
const WordTypeId = 255
const InvoiceMaskNo = 8

Set Elo=CreateObject("Elo.office")

if Elo.SelectView(0) = 1 then
  ParentId = Elo.GetEntryId(-1)
  if ParentId < 1 then
    call Elo.MsgBox("Bitte wählen Sie zuerst einen Zielordner aus", _
                "ELO", vbOkOnly or vbInformation)
  else
    call CreateEntry
  end if
else
  call Elo.MsgBox(_
     "Diese Funktion kann nur in der Archivansicht verwendet werden", _
     "ELO", vbOkOnly or vbInformation)
end if

' Erzeugt an der aktuellen Archivposition eine neue Rechnung
' und ruft Word zur Bearbeitung auf
Sub CreateEntry
  call Elo.PrepareObjectEx(0, WordTypeId, InvoiceMaskNo )
  Elo.ObjIndex = "#" & ParentId
  Elo.ObjShort = "Neue Rechnung"
  if Elo.DoEditObject() = 1 then
    if Elo.UpdateObject() < 0 then
      call Elo.MsgBox(_
        "Das neue Dokument konnte nicht angelegt werden", _
```

```
                "ELO", vbOkOnly or vbInformation)
   else
    Id = Elo.GetEntryId(-2)
    call Elo.LockObject(Id, 1)
    call Elo.GotoId(Id)
    CopyExecTemplateFile(Id)
   end if
  end if
End Sub


' Kopiert die Vorlagendatei in das CheckOut Verzeichnis. Dabei wird
' das spezielle Namensschema beachtet, welches ELO beim CheckIn für
' die Erkennung des Zieldokuments benötigt.
sub CopyExecTemplateFile(Id)
  Dim Source, Dest, fso, file, ArchiveName

  ArchiveName = Replace(Elo.GetArcName(), "_", " ")
  Source = Elo.GetDocumentPath(TemplateDoc, 0)
  Dest = Elo.GetPostDir() & "CheckOut\Rechung_" _
       & ArchiveName & "_" & Id & Elo.SplitFileName(Source, 3)
  Set fso = CreateObject("Scripting.FileSystemObject")
  call fso.CopyFile(Source, Dest)
  Set file = fso.GetFile(Dest)
  file.Attributes = 0
  call Elo.DoExecuteEx(Dest, "", "", "Edit", -1)
end sub
```

Diese Version hat aber noch einen schweren Mangel: wenn der Anwender beim Aufruf nicht auf einem Ordner sondern auf einem Dokument steht, wird das neue Dokument unterhalb des selektierten Dokuments abgelegt. Das ist aus ELO Sicht ein erheblicher struktureller Fehler. Der neue Eintrag kann dann nicht mehr über die Archivstruktur aufgefunden werden, er ist scheinbar verloren. Lediglich über die Suche ist er noch erreichbar. Unter ELOprofessional kann so ein defekter Eintrag zu erheblichen Problemen mit der Replikation führen.

Dieser Mangel lässt sich jedoch recht einfach beheben. Das Ziel des neuen Dokuments darf nicht einfach nur per GetEntryId(-1) bestimmt werden. Damit würde die Verantwortung für eine korrekte Auswahl alleine auf die

Schultern des Anwenders gelegt werden. So eine Vorgehensweise wird im laufenden Betrieb immer wieder zu Fehlern und Irritationen führen. Wenn der Anwender das Skript von einem Dokument aus aufruft, dann gibt es zwei mögliche Strategien:

1. Es wird eine Fehlermeldung ausgegeben, dass der Anwender nur einen Ordner als Ziel angeben darf.
2. Es wird der direkte Vorgängerordner des Dokuments als Ziel verwendet. Das neue Dokument liegt also neben dem aktuell angezeigten Dokument im gleichen Ordner.

Die zweite Variante ist natürlich komfortabler für den Anwender und soll deshalb hier implementiert werden. Dazu wird der Aufruf „ELO.GetEntryId(-1)“ ausgetauscht gegen einen Funktionsaufruf GetParentId(). Diese Funktion führt dann die notwendige Prüfung durch:

```
' Ermittelt aus der aktuellen Archivposition das Ablageziel. Wenn ein
' Ordner aktiv ist, geht das neue Dokument in diesen Ordner. Ist ein
' Dokument aktiv, wird das Vorgängerregister als Ziel verwendet
function GetParentId
  dim ParentId, TreePath

  ParentId = Elo.GetEntryId(-1)
  if ParentId < 1 then
    ParentId = 0
  else
    call Elo.PrepareObjectEx(ParentId, 0, 0)
    if Elo.ObjTypeEx > 253 then
      TreePath = Split(Elo.GetTreePath(1, ",", 255), ",")
      if UBound(TreePath) > 1 then
        ParentId = TreePath(UBound(TreePath) - 1)
      else
        ParentId = 0
      end if
    end if
  end if

  GetParentId = ParentId
end function
```

Hinweis: Dieses Beispiel zeigt, dass man über die OLE Schnittstelle auch Einträge erzeugen kann, die man über das normale Anwenderinterface nicht erstellen könnte. Es gibt hier einige Stellen, an denen keine umfangreichen Kontrollen ausgeführt werden, zum Teil aus Performancegründen, zum Teil einfach aus historischen Gründen und zum Teil ist es anders gar nicht möglich. Es liegt immer in der Verantwortung des Skriptentwicklers, sicher zu stellen, dass nur zulässige Konstruktionen erzeugt werden.

Wiedervorlagetermin zu einem Dokument erzeugen

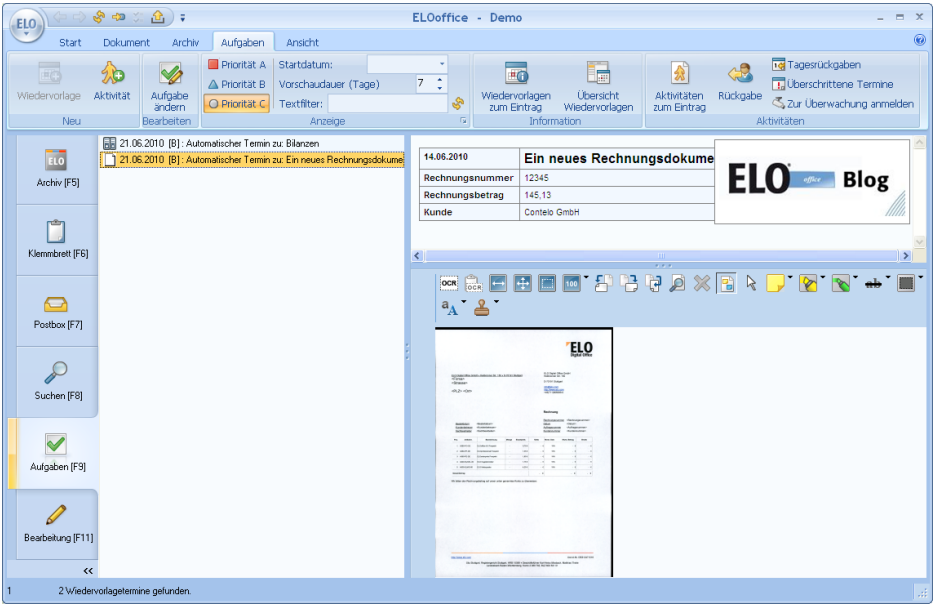
Wiedervorlagetermine sind ein wichtiges Werkzeug, wenn Dokumente zu einem späteren Zeitpunkt noch mal bearbeitet, geprüft oder zur Kenntnis genommen werden müssen. Im Normalfall wird sich ein Anwender bei Bedarf manuell einen Termin erzeugen, das geht mit wenigen Mausklicks. Wenn man Ablagevorgänge aber automatisiert, dann kann es vorkommen, dass auch ein Wiedervorlagetermin automatisch erzeugt werden muss.

Die Erzeugung von Wiedervorlageterminen arbeitet ähnlich wie die der Verschlagwortung. Es muss zuerst mittels ReadWv ein leerer Termin angelegt werden. Dieser wird dann über seine Properties mit Daten gefüllt und durch WriteWv in die Datenbank geschrieben.

Das folgende Skript legt zum aktuell selektierten Eintrag einen neuen Wiedervorlagetermin für den Administrator zum 21.6.2011 an.

```
' Skript: Wiedervorlage erzeugen
' Autor: Matthias Thiele
' Erstellt: 14.06.2010
' Letzte Änderung: 14.06.2010
'
' Dieses Programm erzeugt zum aktuell ausgewählten
' Eintrag einen Wiedervorlagetermin für den Administrator


option explicit

Const AdministratorId = 0
Const NewWV = 0

Dim ELO, Id
Set ELO = CreateObject("ELO.office")

Id = ELO.GetEntryId(-1)
If Id < 1 Then
  Call ELO.MsgBox("Bitte wählen Sie zuerst einen Eintrag aus.", _
              "ELO", vbOkOnly)
Else
  If ELO.PrepareObjectEx(Id, 0, 0) < 0 Then
```

```
    Call ELO.MsgBox("Der Eintrag konnte nicht gelesen werden.", _
                    "ELO", vbOkOnly)
  Else
    Call ELO.ReadWv(NewWV)
    ELO.WvParent = Id
    ELO.WvUserOwner = AdministratorId
    ELO.WvShort = "Automatischer Termin zu: " & ELO.ObjShort
    ELO.WvParentType = ELO.ObjTypeEx
    ELO.WvDate = "21.06.2011"
    Call ELO.WriteWv(NewWV)
  End If
End If
```



Hinweis: Es gibt einige Funktionen in mehreren Varianten, z.B. ObjType und ObjTypeEx. In diesem Fall sollten Sie prinzipiell die *Ex Funktion verwenden, da diese neuer ist und zusätzliche Funktionen unterstützt. Die alte Version ist jeweils nur noch aus Kompatibilitätsgründen vorhanden, damit alte Skripte weiter funktionsfähig bleiben und sollte für Neuentwicklungen nicht verwendet werden.

Funktionen zu Wiedervorlageterminen

Wiedervorlagetermine werden über die Funktionen ReadWv und WriteWv gelesen und geschrieben. Bestehende Termine werden über die Wiedervorlagenummer gelesen, neue Termine mittels der Pseudo-Id 0 erzeugt. Auf den Termin kann dann mit folgenden Properties lesend und schreibend zugegriffen werden:

WvIdent	(ReadOnly) Interne ELO Wiedervorlagenummer des Termins. Diese Nummer wird intern von ELO verwaltet und vergeben. Neue Einträge werden über die Nummer 0 angelegt, das System vergibt dann intern die nächste freie Nummer.
WvParent	ELO Objektnummer des Ordners oder Dokuments zu dem dieser Termin erzeugt wurde.
WvUserFrom	ELO Anwendernummer der Person, die den Termin erzeugt hat. Prinzipiell könnten Sie per Skript unter den Anwender A einen Termin erzeugen, der scheinbar von einem Anwender B angelegt wurde. Das ist im Normalfall aber schlechter Stil und sollte nur in besonders begründeten Situationen verwendet werden.

WvUserOwner	Anwendernummer der Person, die diesen Termin bearbeiten soll.
WvCreateDate	Datum an dem der Termin erzeugt wurde. Dieses Property muss nicht gefüllt werden, es wird dann beim Speichern automatisch auf das aktuelle Tagesdatum gesetzt.
WvDate	Fälligkeitsdatum des Termins.
WvPrio	Priorität des Termins. Es stehen die Werte 1 (Wichtig), 2 (Normal) und 3 (weniger Wichtig) zur Verfügung. Alle anderen Werte werden automatisch auf 2 (Normale Priorität) gesetzt.
WvParentType	Typ des zugeordneten Ordners (1..32) oder Dokuments (254..285). Wenn ein Termin per Skript erzeugt wird, sollten Sie darauf achten, dass Sie diesen Typ aus dem Originaleintrag per GetObjTypeEx übernehmen. Wenn es hier Unterschiede zwischen dem Eintrag und dem Termin gibt, kann das beim Anwender zu erheblichen Irritationen führen.
WvShort	Kurzbezeichnung des Termins. Bei manuell angelegten Terminen wird hier als Vorgabe die Kurzbezeichnung des Ordners oder Dokuments eingetragen. Der Anwender kann diesen Text aber vor dem Speichern beliebig abändern.
WvDesc	Arbeitsanweisung zu dem Termin.

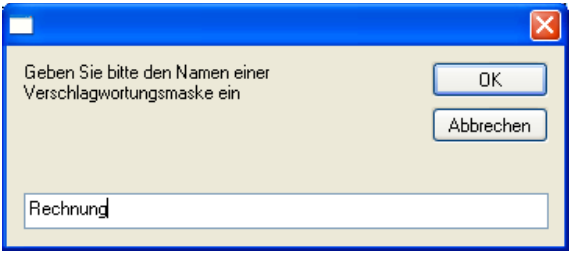
Arbeiten mit Verschlagwortungsmasken

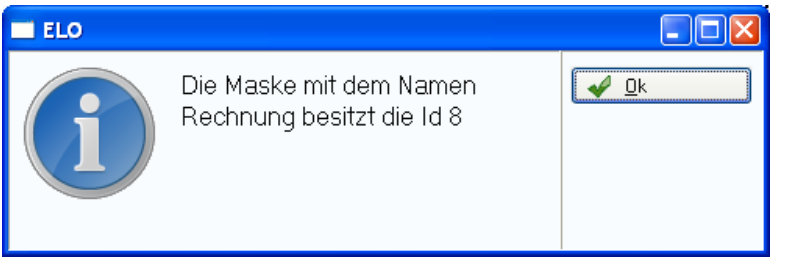
Maskennummer aus dem Namen ermitteln

In den letzten Beispielen wurde mehrfach mit der Verschlagwortungsmaskennummer gearbeitet. In einem Archiv ist es oft sinnvoll direkt mit der Nummer zu arbeiten (z.B. 0 für „Freie Eingabe“). Diese kann man über den Verschlagwortungsmaskendialog ermitteln. Wenn man Skripte weitergeben möchte oder in unterschiedlichen Archiven verwenden möchte, ist es oft nicht möglich, mit einer festen Nummer zu arbeiten. Diese ist dann entweder unterschiedlich oder gar nicht bekannt. Ein Ausweg aus dieser Situation besteht darin, dass man mit dem Verschlagwortungsmaskennamen statt der Nummer arbeitet. Die Nummer zu einer Maske kann per LookupMaskName ermittelt werden.

```
' Skript: Maskennummer ermitteln
' Autor: Matthias Thiele
' Erstellt: 14.06.2010
' Letzte Änderung: 14.06.2010
'
' Dieses Programm fragt beim Anwender den Namen einer
' Verschlagwortungsmaske ab und gibt die Nummer dazu aus


Option Explicit

Dim ELO, Name, Id
Set ELO = CreateObject("ELO.office")

Name = InputBox("Geben Sie bitte den Namen " _
            & "einer Verschlagwortungsmaske ein")
If Name <> "" Then
  Id = ELO.LookupMaskName(Name)
  If Id < 0 then
    Call ELO.MsgBox("Eine Maske mit dem Namen " & Name _
            & " ist im System nicht bekannt", _
            "ELO", vbOkOnly)
  Else
```

```
    Call ELO.MsgBox("Die Maske mit dem Namen " & Name _
            & " besitzt die Id " & Id, _
            "ELO", vbOkOnly)
  End If
End If
```



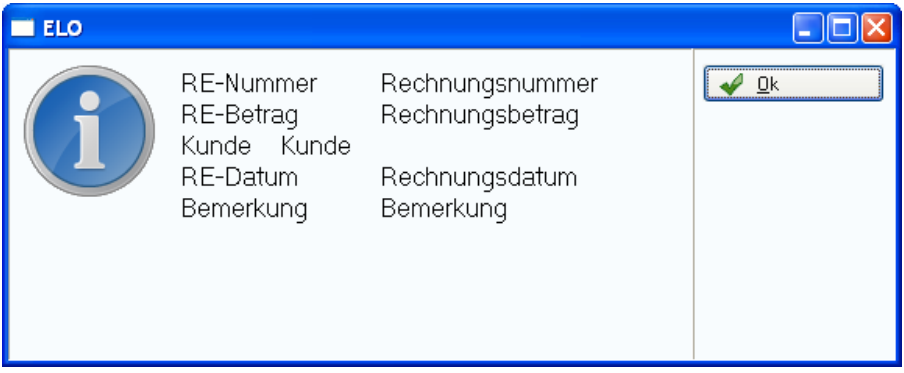


Verschlagwortungsmaskeneinstellungen lesen

Auch ohne Dokument oder Ordner können Sie die Einstellungen zu einer Verschlagwortungsmaske lesen. Dafür gibt es den Befehl ReadObjMask. Das folgende Beispiel liest die Ablagemaske 8 (im Demo Archiv ist das die Rechnungsmaske) ein und listet alle definierten Indexzeilen auf.

```
' Skript: Indexzeilen auflisten
' Autor: Matthias Thiele
' Erstellt: 14.06.2010
' Letzte Änderung: 14.06.2010
'
' Dieses Programm listet zu einer definierten
' Verschlagwortungsmaskennummer alle Indexzeilen auf


Option Explicit
```

```
Const InvoiceMaskNo = 8

Dim ELO, Name, i, Key, Message
Set ELO = CreateObject("ELO.office")

If ELO.ReadObjMask(InvoiceMaskNo) < 0 Then
  Call ELO.MsgBox("Es gibt keine Verschlagwortungsmaske " _
         & "mit der Nummer " & InvoiceMaskNo, "ELO", vbOkOnly)
Else
  For i = 0 to 49
    Key = ELO.GetObjAttribKey(i)
    If Key <> "" Then
      Name = ELO.GetObjAttribName(i)
      Message = Message & Key & vbTab & Name & vbCrLf
    End If
  Next
  Call ELO.MsgBox(Message, "ELO", vbOkOnly)
End If
```



Hinweis: Prinzipiell kann man Verschlagwortungsmasken auch über ein Skript verändern und speichern (WriteObjMask). Wenn man sich nicht sehr gut im System auskennt, sollte man unbedingt auf diese Möglichkeit verzichten. Eine ungeeignete Veränderung im laufenden Betrieb kann dazu führen, dass die bereits vorhandenen Einträge dieser Maske nicht mehr korrekt angezeigt werden.

Datenbankzugriff mittels ADODB

In der Praxis kommt es oft vor, dass man aus einem Skript heraus auf andere Datenbestände zugreifen muss. Das können Kundendaten aus der Verwaltung sein, Produktnamen aus dem Lager oder auch Kalenderdaten.

Erfreulicherweise können Sie mittels VB-Script und ADODB relativ leicht auf beliebige Datenbanken zugreifen. Der prinzipielle Ablauf sieht so aus:

1. Es wird ein ADODB Objekt mittels CreateObject erzeugt.
2. Open() öffnet eine Verbindung zur Datenbank.
3. Über den Befehl Execute() kann anschließend eine Abfrage gestartet werden. Diese Abfrage liefert als Ergebnis einen ResultSet mit der Trefferliste.
4. Aus der Trefferliste können nun die Daten ausgelesen werden. Nach der Abfrage ist der erste Treffer aktiv, weitere Treffer können dann jeweils mit MoveNext abgerufen werden.
5. Am Ende wird der ResultSet mit Close wieder geschlossen.

Der Kernbereich so einer Abfrage sieht dann so aus:

```
Set ResultSet = DB.Execute(SqlCommand)

Do Until ResultSet.EOF
  List=List & ResultSet(0) & vbCrLf
  ResultSet.MoveNext
Loop

ResultSet.Close
```

Hinweis: Unter VB-Script sehen Sie manchmal direkte Zuweisungen „Maske = 5“ und in anderen Fällen eine Variante mit vorangestellten Set: Set ELO = CreateObject(…). Das liegt daran, dass Sie nur einfache Datentypen über die einfache Form direkt zuweisen können. Wenn Sie mit Objekten arbeiten, dann wird das zusätzliche Set benötigt.

Eine ODBC Datenbankabfrage

Ein vollständiges Skript, welches in der ELO Demo Datenbank nach Einträgen mit „ELO“ in der Kurzbezeichnung sucht, sieht dann so aus:

```
' Skript: ODBC Datenbankabfrage
' Autor: Matthias Thiele
' Erstellt: 11.06.2010
' Letzte Änderung: 13.06.2010
'
' Dieses Programm führt eine Datenbankabfrage
' über eine ADO Verbindung aus.

option explicit

const SqlCommand = "Select objid, objtstamp, objshort from objekte
where objshort like '%elo%'"

Dim ELO, DB, ResultSet, List, i
Set ELO = CreateObject("ELO.office")
Set DB = CreateObject("ADODB.Connection")

' Anmeldung - ODBC Connector, Anwender, Passwort
Call DB.Open("Elo32o_Demo", "", "" )

' Suchanfrage absetzen
Set ResultSet = DB.Execute(SqlCommand)

' und das Ergebnis einlesen
Do Until ResultSet.EOF
  For i = 0 to 2
    List=List & ResultSet(i) & vbTab
  Next
  List=List & vbCRLF
  ResultSet.MoveNext
Loop

ResultSet.Close

' Zuletz noch die gefundenen Einträge anzeigen
call ELO.MsgBox(List, "ELO", vbOkOnly)
```

Hinweis: In diesem Beispiel wurde direkt auf die ELO Datenbank zugegriffen. In produktiven Umgebungen sollten Sie das nicht machen, da sich die Datenbankstruktur jederzeit ändern könnte und Ihr Skript dann nicht mehr korrekt arbeiten würde.

Eine Excel Datenbankabfrage

Mittels ADODB können Sie auch Excel Dokumente auslesen. Das folgende Beispiel demonstriert es anhand eines Excel Dokuments aus der ELO Demo Datenbank. Falls Sie nicht mit dem Demo Archiv arbeiten oder eine ältere Version des Demo Archivs einsetzen, müssen Sie in der Konstantendefinition die Eintragsnummer der Excel DocumentId anpassen, im Demo Archiv lautet sie 10.

Der eigentliche Programmablauf ist ähnlich zum Beispiel zuvor. Statt einer ADODB Connection wird nun aber direkt ein ADODB ResultSet geöffnet.

```
Set ResultSet = CreateObject("ADODB.Recordset")
...
Conn = "DRIVER=Microsoft Excel-Treiber (*.xls);DBQ=" & ExcelPath
Call ResultSet.Open(SqlCommand, Conn, 3, 3 )
```

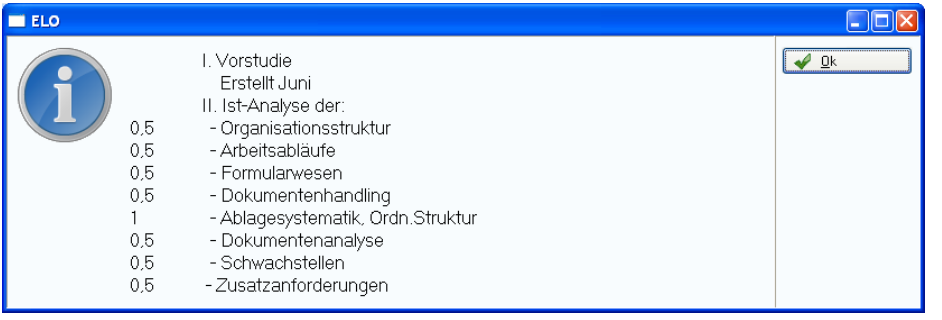
Für den Zugriff auf die Excel Datei wird zuerst aus der Objektnummer 10 der Dateipfad und Dateiname ermittelt. Hierfür wird der Befehl GetDocumentPath verwendet.

```
ExcelPath = ELO.GetDocumentPath(ExcelDocumentId, 0)
If ExcelPath = "" Then
  Call ELO.MsgBox("Die Excel Datei konnte nicht eingelesen werden.",
"ELO", vbOkOnly)
Else
  call ReadExcel(ExcelPath)
End if
```

Das vollständige Skript sieht dann so aus:

```
' Skript: Excel Abfrage
' Autor: Matthias Thiele
' Erstellt: 14.06.2010
' Letzte Änderung: 14.06.2010
'
' Dieses Programm liest aus einem Excel Dokument des Demo
' Archivs verschiedene Daten aus

option explicit

const SqlCommand = "SELECT * FROM [Tabelle1$a1:c]"
const ExcelDocumentId = 10

Dim ELO, ExcelPath, ResultSet
Set ELO = CreateObject("ELO.office")
Set ResultSet = CreateObject("ADODB.Recordset")

ExcelPath = ELO.GetDocumentPath(ExcelDocumentId, 0)
If ExcelPath = "" Then
  Call ELO.MsgBox(_
    "Die Excel Datei konnte nicht eingelesen werden.", _
    "ELO", vbOkOnly)
Else
MsgBox ExcelPath
  call ReadExcel(ExcelPath)
End if

Sub ReadExcel(ExcelPath)
  Dim Conn, i, List
  Conn = "DRIVER=Microsoft Excel-Treiber (*.xls);DBQ=" & ExcelPath

  ' Abfrage - Connection - CursorType - LockType
  Call ResultSet.Open(SqlCommand, Conn, 3, 3 )
```

```
  ' und das Ergebnis einlesen
  Do Until ResultSet.EOF
    List = List & ResultSet.Fields(1) & vbTab
    List = List & ResultSet.Fields(0) & vbCRLF
    ResultSet.MoveNext
  Loop

  ResultSet.Close

  ' Zuletz noch die gefundenen Einträge anzeigen
  call ELO.MsgBox(List, "ELO", vbOkOnly)
End Sub
```



Archivbaum durchlaufen

Bei Automatisierungsaufgaben gibt es häufig die Anforderung, dass ein kompletter Archivzweig durchlaufen werden muss und zu jedem Untereintrag eine Aktion ausgeführt werden soll. Diese Aufgabe kann man auf zwei Arten realisieren:

1. **Über den OLE Befehl TreeWalk:** Dieser Aufruf ist für den Skriptentwickler sehr einfach. Sie geben einen Startpunkt an und ein Skript, welches für jeden Eintrag aufgerufen werden soll. Um den Durchlauf kümmert sich das ELOoffice OLE Automation System. In dem Aufrufskript können Sie sich jeweils auf die Aktionen für diesen Eintrag konzentrieren. Der Nachteil dieser Variante liegt darin, dass jeder Skriptaufruf nur für einen einzelnen Eintrag gilt. Es ist relativ aufwändig, Informationen zwischen den einzelnen Skriptaufrufen zu übertragen.
2. **Der Baum kann auch manuell im Skript durchlaufen werden:** Dazu können für jeden Knoten die Nachfolgerlisten per CollectChildList abgerufen werden. Dieser Weg erfordert eine umfangreichere Skript Programmierung, er hat aber den Vorteil, dass die komplette Abarbeitung von einem Skript durchgeführt werden kann. Somit ist es leichter, Informationen zwischen den einzelnen Objekten auszutauschen.

Hinweis: Von einer dritten Möglichkeit, per GotoId einen Ordner aufzurufen und dann über GetEntryId die Nachfolgerliste abzulaufen und für jeden Nachfolger dann ein erneutes GotoId aufzurufen, raten wir ab. Dieser Weg führt zu einer umfangreichen Anzeigeaktivität und kann bei sehr großen Archivbäumen zu einer extremen Speicherauslastung bis hin zum Programmabsturz führen.

Anwendung des TreeWalk Befehls

Dieses erste Beispiel durchläuft einen Teilbaum unter dem aktuell selektierten Eintrag und setzt den Farbmarker für jedes Objekt auf Rot, wenn die Kurzbezeichnung den Text ‚ELO' enthält. Da hier zwischen den Einträgen keine Informationen übertragen werden müssen, bietet sich der Einsatz des TreeWalk Befehls an. Es müssen also zwei Skripte erstellt werden: das Hauptskript, welches die aktuelle Archivposition ermittelt und den TreeWalk Befehl aufruft. Und ein Callback Skript, welches für jeden Eintrag aufgerufen wird, die Kurzbezeichnung prüft und bei Bedarf die Farbe setzt.

Das Hauptskript ist sehr einfach. Es wird über GetEntryId(-1) die aktuelle Archivposition abgefragt und anschließend der TreeWalk Befehl gestartet.

```
Option Explicit

Const WalkOrder = 1
Const MaxLevel = 15

Dim ELO, StartId
Set ELO = CreateObject("ELO.office")

StartId = ELO.GetEntryId(-1)
If StartId > 0 Then
  Call ELO.TreeWalk(WalkOrder, StartId, MaxLevel, "ZZZ_UpdateColor")
End
```

Das Callback Skript hat zur besseren Erkennbarkeit den gleichen Namen wie das Hauptskript, jedoch mit einem vorangestellten ZZZ_. Was für ein Präfix man verwendet, ist dem Anwender frei gestellt. Da die Callback Skripte aber nicht direkt aufgerufen werden sollen, ist es sinnvoll, sie durch eine geeignete Benennung ganz an das Ende der Skriptliste zu stellen.

```
Option Explicit

Const ColorRed = 1

Dim ELO
Set ELO = CreateObject("ELO.office")
```

```
If (Instr(ELO.ObjShort, "ELO") > 0) _
   and (ELO.DocKind <> ColorRed) Then
  ELO.DocKind = ColorRed
  Call ELO.UpdateObject
End If
```

Der TreeWalk Befehl führt vor dem Aufruf des Callback Skripts intern ein PrepareObjectEx aus. Es sind dann alle Verschlagwortungsdaten geladen und sie können direkt verwendet werden. Deshalb kann direkt die Belegung des ObjShort Feldes (Kurzbezeichnung) geprüft werden. Da aber nicht jeder Eintrag verändert wird, muss das Callback Skript einen möglicherweise notwendigen Speichervorgang durch UpdateObject selber auslösen. In diesem Fall ist das nur sinnvoll, wenn die Farbe geändert wurde.

Über den Mode Parameter des TreeWalk Befehls kann man steuern, ob das Callback Skript für die Nachfolgerknoten vor dem Startknoten, nach dem Startknoten oder zu beiden Zeitpunkten aufgerufen werden soll. Wenn die Bearbeitung der Einträge unabhängig voneinander ist (wie in diesem Beispiel), dann ist der Zeitpunkt des Aufrufs unwichtig. Wenn in der Bearbeitung die Verschlagwortung des Startknotens auf die Nachfolger übertragen werden soll, dann muss zuerst der Startknoten bearbeitet werden, danach die Nachfolger. Und wenn der Startknoten eine Summe eines Indexfeldes der Nachfolger erhalten soll, dann müssen zuerst die Nachfolger und danach der Startknoten bearbeitet werden.

Skriptgesteuerter Tree Walk

In diesem Abschnitt wird die gleiche Funktion wie zuvor gezeigt, nur statt des TreeWalk Befehls wird ein skriptgesteuerter Durchlauf erzeugt. In diesem einfachen Beispiel hat man dadurch noch keinen Vorteil, wir werden aber später noch in einem etwas komplexeren Umfeld Nutzen daraus ziehen können.

Der Ablauf des Skripts sieht so aus, dass es eine Walk Subroutine gibt mit einem Parameter für den aktuellen Eintrag. Diese bearbeitet zuerst den aktuellen Eintrag, dazu muss dieser mittels PrepareObjectEx eingelesen werden. Falls es sich bei dem Eintrag um einen Ordner handelt, wird anschließend die Liste der Nachfolger eingelesen und für jeden Untereintrag wiederum rekursiv die Funktion Walk aufgerufen.

```
Option Explicit

Const ColorRed = 1
Const MaxFolder = 32

Dim ELO, StartId
Set ELO = CreateObject("ELO.office")

StartId = ELO.GetEntryId(-1)
If StartId > 0 Then
  Call Walk(StartId)
End If


Sub Walk(StartId)
  Dim ChildListString, ChildList, i, Msg

  ' Zuerst den Eintrag selber einlesen und prüfen
  If ELO.PrepareObjectEx(StartId, 0, 0) < 0 Then
    ' Der Eintrag konnte nicht gelesen werden, hier abbrechen
    Exit Sub
  End If

  If Instr(ELO.ObjShort, "ELO") > 0 Then
    ELO.DocKind = ColorRed
    Call ELO.UpdateObject
  End If

  ' Nun die Nachfolgerliste prüfen. Nur Ordner haben Nachfolger
  If ELO.ObjTypeEx <= MaxFolder Then
    ChildListString = ELO.CollectChildList(StartId)
    If ChildListString <> "" Then
      ChildList = Split(ChildListString, ":")
      For i = 0 to UBound(ChildList)
        If ChildList(i) <> "" Then
          Call Walk(ChildList(i))
```

```
      End If
    Next
  End If
End If
End Sub
```

Dieser Code funktioniert zwar, er hat aber zwei Schönheitsfehler:

1. Obwohl dieser Baumdurchlauf eine häufig benötigte Operation ist, kann der Skript Code nur schlecht wiederverwendet werden, da der allgemeine Durchlauf Code mit dem speziellen Bearbeitungscode vermischt wurde.
2. Wenn der Baum zyklische Referenzen besitzt (A ist in B und B in C und C in A), dann läuft das Skript endlos im Kreis bis der komplette Speicher verbraucht ist. Ideal wäre eine Erkennung bereits betretener Knoten, mindestens sollte es aber eine maximale Ebenentiefe geben (so wie beim TreeWalk Befehl).

Eine TreeWalk Klasse in VB-Script

Im Folgenden soll das vorhandene Beispiel weiter ausgebaut werden. Zum einen wird eine maximale Ebenentiefe bestimmt, die vermeidet, dass sich der Lauf in einer Schleife fängt. Zum anderen wird der Code des Baumdurchlaufs vom Code zur Veränderung der Einträge getrennt.

In einer vollständigen OOP Umgebung würde man hier eine Klasse schaffen, die zwei getrennte Methoden besitzt, einmal eine „Process“ Methode, die den Code für die Änderungen enthält und einmal eine „Walk“ Methode, die den eigentlichen Baum Durchlauf verwaltet. Für jeden Eintrag wird dann aus der Walk Methode heraus die Process Methode aufgerufen. Die Basisklasse hat dann eine leere Process Methode. Für die Lösung eines Problems wird von der Basisklasse eine neue Klasse abgeleitet und die Prozessmethode mit dem notwendigen Code überschrieben.

In VB-Script geht das leider nicht. Deshalb muss man die Process Methode als externen Methodenaufruf verwalten. Der Rest kann dann über eine ScriptWalk Klasse abgebildet werden. Ein erster Ansatz würde so aussehen, dass man eine globale Funktion „Process_TreeWalk“ anlegen würde. Diese Vorgehensweise hätte aber den Nachteil, dass man nur eine Verwendung dieser Klasse pro Skript Modul erlauben könnte, da es ja nur einen Funktionsnamen gibt.

Glücklicherweise bietet VB-Script aber die Möglichkeit Funktions-Zeiger als Parameter zu übergeben. In diesem Fall kann man eine Funktion mit einem beliebigen Namen für die Ausführung anlegen und anschließend diese Funktion dem Walk Befehl zur Ausführung übergeben.

Ein kurzes Beispiel definiert zwei unterschiedliche Ausgabefunktionen Msg1 und Msg2. Die Verarbeitungsfunktion Test wird mal mit der einen und mal in der Anderen Ausgabe aufgerufen.

```
Dim FP_Msg1, FP_Msg2

Set FP_Msg1 = GetRef("Msg1")
Set FP_Msg2 = GetRef("Msg2")

Test(FP_Msg1)
Test(FP_Msg2)

Sub Msg1
  MsgBox "Das ist eine Nachricht"
End Sub

Sub Msg2
  MsgBox "Das ist eine andere Nachricht"
End Sub

Sub Test(FP_Message)
  Call FP_Message
End Sub
```

Prinzipiell könnte das ScriptWalk Modul etwa so aussehen (gekürzter, unvollständiger Code):

```
Call MyScriptWalk.Walk(StartId, ScriptWalk_Process)
…

Sub ScriptWalk_Process
  …
End Sub

Class ScriptWalk
  Private Process

  Private Sub Walk(StartId, Callback)
    Set Process = Callback
    …

    Call Process

    …
      For i = 0 to UBound(ChildList)
         If ChildList(i) <> "" Then
           Call Walk(ChildList(i))
         End If
       Next
    …
  End Sub
End Class
```

Die Subroutine „Process“ ist keine Klassenmethode, sie muss extern vom Skript geliefert werden.

Die Änderungen an den Einträgen werden über diese Process Funktion durchgeführt:

```
Sub ScriptWalk_Process
    If Instr(ELO.ObjShort, "ELO") > 0 Then
      ELO.DocKind = ColorRed
      Call ELO.UpdateObject
    End If
End Sub
```

Wenn man nun ein einem Skript einen TreeWalk benötigt, muss man nur die Klasse ScriptWalk in die Skriptdatei kopieren und eine Subroutine ScriptWalk_Process implementieren.

Hinweis: OOP Puristen werden vermutlich einwenden, dass es ein Designfehler ist, dass die ScriptWalk Klasse die Existenz der globalen Variablen „ELO“ voraussetzt. Wenn man das als störend empfindet, dann kann man diese Variable als Parameter an die Klasse übergeben. In normalen ELO Skripten ist diese Variable aber immer notwendig, so dass es unkritisch ist, die Existenz vorauszusetzen.

Das komplette Skript sieht dann so aus:

```
' Skript: ScriptWalk2
' Autor: Matthias Thiele
' Erstellt: 15.06.2010
' Letzte Änderung: 28.06.2010
'
' Dieses Skript läuft mittels der Funktion CollectChildList
' über alle Nachfolger eines Starteintrags und setzt die
' Farbe auf Rot wenn die Kurzbezeichnung den Text ELO enthält.


Option Explicit

Const ColorRed = 1
Const MaxFolder = 32

Dim ELO, StartId, ColorUpdate, Callback
Set ELO = CreateObject("ELO.office")

StartId = ELO.GetEntryId(-1)
If StartId > 0 Then
  Set Callback = GetRef("ScriptWalk_Process")
  Set ColorUpdate = new ScriptWalk
  Call ColorUpdate.Walk(StartId, MaxFolder, Callback)
End If

Sub ScriptWalk_Process
  If Instr(ELO.ObjShort, "ELO") > 0 Then
    ELO.DocKind = ColorRed
```

```
    Call ELO.UpdateObject
  End If
End Sub

Class ScriptWalk
  Private Level
  Private LocalMaxLevel
  Private ActPath
  Private ProcessFunction
  Private M_LevelName

  Public Sub Walk(StartId, MaxLevel, Process)
    Level = 0
    ActPath = ""
    LocalMaxLevel = MaxLevel
    Set ProcessFunction = Process

    Call LocalWalk(StartId)
  End Sub

  Public Function GetActPath
    GetActPath = ActPath
  End Function

  Public Property Get LevelName
    LevelName = M_LevelName
  End Property

  Public Property Let LevelName(NewName)
    M_LevelName = NewName
  End Property

  Private Sub LocalWalk(StartId)
    Dim ChildListString, ChildList, i, Msg, OldPath

    ' Zuerst den Eintrag selber einlesen und prüfen
    If ELO.PrepareObjectEx(StartId, 0, 0) < 0 Then
      ' Der Eintrag konnte nicht gelesen werden, hier abbrechen
      Exit Sub
    End If

    M_LevelName = ELO.ObjShort
    Call ProcessFunction

    ' Nun die Nachfolgerliste prüfen. Nur Ordner haben Nachfolger
```

```
    If ((ELO.ObjTypeEx <= MaxFolder) Or (ELO.ObjTypeEx = 9999))_
        and (Level < LocalMaxLevel) Then
      ChildListString = ELO.CollectChildList(StartId)
      If ChildListString <> "" Then
        ChildList = Split(ChildListString, ":")
        Level = Level + 1
        OldPath = ActPath
        ActPath = ActPath & "\" & M_LevelName
        For i = 0 to UBound(ChildList)
          If ChildList(i) <> "" Then
            Call LocalWalk(ChildList(i))
          End If
        Next
        Level = Level - 1
        ActPath = OldPath
      End If
    End If
  End Sub
End Class
```

Die Walk Methode nimmt nun neben dem Startknoten eine maximale Ebenentiefe und den Namen der Callback Routine für die Aktionen mit auf. Sie speichert diese Werte in privaten Variablen und ruft dann eine lokale Walk Methode für den eigentlichen Durchlauf auf. Dort wird bei der Rekursion die aktuelle Ebenentiefe mit der maximalen Tiefe verglichen und beim Erreichen der Grenze nicht weiter in die Tiefe verzweigt.

Die Klasse ScriptWalk enthält nun keinen Code mehr zur Veränderung der Einträge, sie kann deshalb ganz allgemein eingesetzt werden. Damit die Process Methode Zugriff auf den aktuellen Archivpfad besitzt, wurde eine öffentliche Methode GetActPath hinzugefügt. Diese gibt den aktuell aktiven Teilpfad zurück. Im Normalfall wird dieser Pfad automatisch aus der Kurzbezeichnung gebildet. Über das Property LevelName kann die Process Subroutine den Namen aber beeinflussen, z.B. um unerwünschte Zeichen herauszufiltern.

Export eines Teilarchivs in das Dateisystem

Die im letzten Beispiel erzeugte ScriptWalk Klasse soll nun zum Aufbau eines Dokumentenexporters verwendet werden. Dabei soll sich die ELO Archivstruktur im Dateisystem als Ordnerstruktur wiederspiegeln. Ein ELO Ordner wird zum Dateisystemordner, ein ELO Dokument zur Datei. Die ELO Kurzbezeichnung wird jeweils als Ordner oder Dateiname verwendet.

Für dieses Beispiel muss eine passende ScriptWalk_Prozess Subroutine implementiert werden. Im einfachsten Fall sieht sie so aus:

```
Sub ScriptWalk_Process
  Dim Source, Destination, Ext, Name

  Exporter.LevelName = PrepareFilePath(Exporter.LevelName)

  If ELO.ObjTypeEx < MaxFolder Then
    ' Ordner anlegen
    Destination = DestinationBase & Exporter.GetActPath & _
        "\" & Exporter.LevelName
    Call FSO.CreateFolder(Destination)
  Else
    ' Dokument kopieren
    Source = ELO.GetDocumentPath(ELO.GetEntryId(-2), 0)
    If Source <> "" Then
      Ext = ELO.SplitFileName(Source, 3)
      Destination = DestinationBase & Exporter.GetActPath & _
          "\" & Exporter.LevelName & Ext
      Call FSO.CopyFile(Source, Destination)
    End If
  End If
End Sub
```

Wenn der aktuelle Eintrag ein Ordner ist, wird der Ordnername ermittelt und mit CreateFolder ein entsprechender Ordner im Dateisystem angelegt. Handelt es sich um ein Dokument, wird die ELO Datei ermittelt und in das Dateisystem kopiert. Dabei wird an die Kurzbezeichnung aber noch zusätzlich die Dateikennung (Extension) kopiert, damit die Datei benutzbar bleibt. In beiden Fällen werden vorher aus der Kurzbezeichnung einige kritische

Zeichen (Doppelpunkt, Größer und Prozent) entfernt, da diese in Dateinamen zu massiven Problemen führen können. Dazu wird das Property LevelName für jeden Eintrag über die Funktion PrepareFilePath so abgeändert, dass kritische Zeichen gegen einen Unterstrich ersetzt werden. Lassen Sie sich an dieser Stelle nicht von dem regulären Ausdruck RegExp erschrecken. Die Verwendung wird in einem späteren Kapitel genauer erklärt. Im Augenblick reicht es zu wissen, dass Zeichen wie * : \ & | und Tabs und Zeilenwechsel gegen einen Unterstrich ersetzt werden. Im Prinzip verhält sich dieser Teil wie eine Kette von Replace(FilePath, „*“, „_“) Ausdrücken.

```
Function PrepareFilePath(FilePath)
  Dim Reg
  Set Reg = New RegExp

  Reg.Global = True
  Reg  Reg.Pattern = "[\*\:\\\&\t\|\r\n<>]"
  PrepareFilePath = Reg.Replace(FilePath, "_")
End Function
```

Hinweis: das Skript verwendet einen fest voreingestellten Zielordner (C:\Temp\Export in DestinationBase). Dieser muss vor der Ausführung angelegt werden und sollte leer sein. Fall ein anderes Ziel verwendet werden soll, muss das Skript entsprechend abgeändert werden.

Zusammen mit der bereits vorhandenen ScriptWalk Klasse ergibt sich dann folgendes Skript:

```
' Skript: DokumentenExport
' Autor: Matthias Thiele
' Erstellt: 15.06.2010
' Letzte Änderung: 28.06.2010
'
' Dieses Skript läuft mittels der Klasse ScriptWalk
' über alle Nachfolger eines Starteintrags und kopiert
' die Archivstruktur ins Dateisystem.
```

```
Option Explicit

Const DestinationBase = "C:\Temp\Export"
Const MaxFolder = 32
Const ArchiveStartNode = 9999

Dim ELO, FSO, StartId, Exporter, Callback
Set ELO = CreateObject("ELO.office")
Set FSO = CreateObject("Scripting.FileSystemObject")

StartId = ELO.GetEntryId(-1)
If StartId = 0 Then StartId = 1
If StartId > 0 Then
  Set Callback = GetRef("ScriptWalk_Process")
  Set Exporter = new ScriptWalk
  Call Exporter.Walk(StartId, MaxFolder, Callback)
  Call ELO.Status("")
End If


Sub ScriptWalk_Process
  Exporter.LevelName = PrepareFilePath(Exporter.LevelName)
  call ELO.Status(Exporter.LevelName)

  If (ELO.ObjTypeEx < MaxFolder) Or _
     (ELO.ObjTypeEx = ArchiveStartNode) Then
    CreateFolder
  Else
    CopyFile
  End If
End Sub


Sub CreateFolder
  Dim Destination, Name, ActPath
  ActPath = Exporter.GetActPath
  Name = LookupFolderName(DestinationBase, _
         ActPath, Exporter.LevelName)
  Destination = BuildName(DestinationBase, ActPath, Name, "")
  Exporter.LevelName = Name

  On Error Resume Next
  Call FSO.CreateFolder(Destination)
  If Err.Number > 0 Then
    call ELO.MsgBox(Err.Number & " " & Err.Description _
```

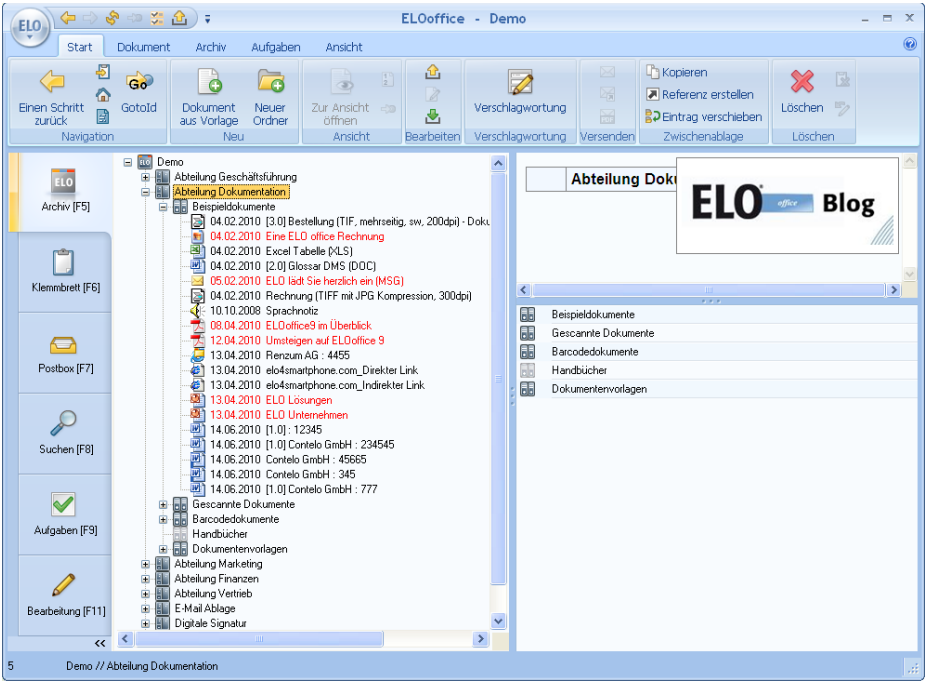
```
          & vbCrLf & Destination, "ELO", vbOkOnly)
  End If
End Sub


Sub CopyFile
  Dim Source, Destination, Name, Ext, ActPath
  Source = ELO.GetDocumentPath(ELO.GetEntryId(-2), 0)
  If Source <> "" Then
    Ext = ELO.SplitFileName(Source, 3)
    ActPath = Exporter.GetActPath
    Name = LookupFileName(DestinationBase, ActPath, _
          Exporter.LevelName, Ext)
    Destination = BuildName(DestinationBase, ActPath, Name, Ext)
    Exporter.LevelName = Name

    On Error Resume Next
    Call FSO.CopyFile(Source, Destination)
    If Err.Number > 0 Then
      call ELO.MsgBox(Err.Number & " " & Err.Description _
            & vbCrLf & Destination, "ELO", vbOkOnly)
    End If
  End If
End Sub


Function LookupFolderName(Base, ActPath, FolderName)
  Dim Name, Retry, Destination

  Destination = BuildName(Base, ActPath, FolderName, "")

  Retry = 1
  Name = FolderName
  While FSO.FolderExists(Destination)
    FolderName = Name & "(" & Retry & ")"
    Destination = BuildName(Base, ActPath, FolderName, "")
    Retry = Retry + 1
  Wend
  ELO.DebugOut("New Folder: " & Destination)
  ELO.DebugOut("New Name: " & FolderName)

  LookupFolderName = FolderName
End Function
```

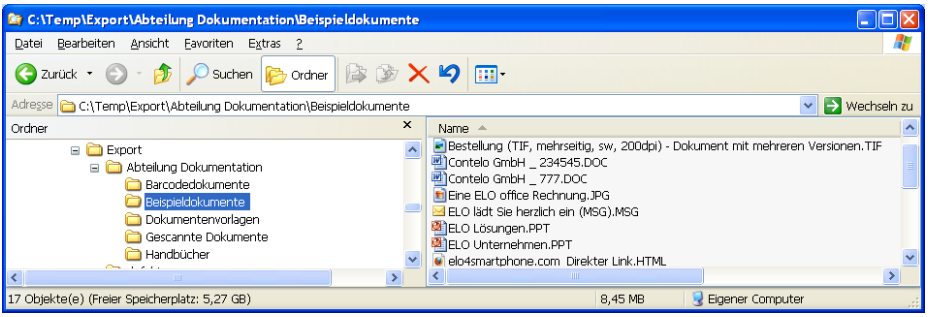
```
Function LookupFileName(Base, ActPath, FileName, Ext)
  Dim Name, Retry, Destination

  Destination = BuildName(Base, ActPath, FileName, Ext)

  Retry = 1
  Name = FileName
  While FSO.FileExists(Destination)
    FileName = Name & "(" & Retry & ")"
    Destination = BuildName(Base, ActPath, FileName, Ext)
    Retry = Retry + 1
  Wend
  ELO.DebugOut("New File: " & Destination)
  LookupFileName = FileName
End Function


Function BuildName(Base, Path, Name, Ext)
  BuildName = Base & Path & "\" & Name & Ext
End Function


Function PrepareFilePath(FilePath)
  Dim Reg
  Set Reg = New RegExp

  Reg.Global = True
  Reg.Pattern = "[\*\:\\\&\t\|\r\n<>]"
  PrepareFilePath = Reg.Replace(FilePath, "_")
End Function


Class ScriptWalk
  Private Level
  Private LocalMaxLevel
  Private ActPath
  Private ProcessFunction
  Private M_LevelName

  Public Sub Walk(StartId, MaxLevel, Process)
    Level = 0
    ActPath = ""
    LocalMaxLevel = MaxLevel
    Set ProcessFunction = Process
```

```
  Call LocalWalk(StartId)
End Sub

Public Function GetActPath
  GetActPath = ActPath
End Function

Public Property Get LevelName
  LevelName = M_LevelName
End Property

Public Property Let LevelName(NewName)
  M_LevelName = NewName
End Property

Private Sub LocalWalk(StartId)
  Dim ChildListString, ChildList, i, Msg, OldPath

  ' Zuerst den Eintrag selber einlesen und prüfen
  If ELO.PrepareObjectEx(StartId, 0, 0) < 0 Then
    ' Der Eintrag konnte nicht gelesen werden, hier abbrechen
    Exit Sub
  End If

  M_LevelName = ELO.ObjShort
  Call ProcessFunction

  ' Nun die Nachfolgerliste prüfen. Nur Ordner haben Nachfolger
  If ((ELO.ObjTypeEx <= MaxFolder) Or (ELO.ObjTypeEx = 9999)) _
      and (Level < LocalMaxLevel) Then
    ChildListString = ELO.CollectChildList(StartId)
    If ChildListString <> "" Then
      ChildList = Split(ChildListString, ":")
      Level = Level + 1
      OldPath = ActPath
      ActPath = ActPath & "\" & M_LevelName
      For i = 0 to UBound(ChildList)
        If ChildList(i) <> "" Then
          Call LocalWalk(ChildList(i))
        End If
      Next
      Level = Level - 1
      ActPath = OldPath
    End If
  End If
```

```
End Sub
End Class
```

Die Wiederverwendung der einmal erstellten ScriptWalk Klasse hat in diesem Beispiel dazu geführt, dass mit 30 Zeilen Skript Code ein einfacher Dokumentenexporter erstellt werden konnte. Hier zeigen sich die Vorteile, wenn man sich eine Sammlung von Hilfsfunktionen aufbaut. Die Verwendung von Klassen statt einzelner Funktionen vermeidet dabei Namenskollisionen und erlaubt die Verwendung geschützter lokaler Variablen. Durch die Verwendung von Callback Funktionen mittels Funktionszeiger konnte in diesem Beispiel auch die fehlende Vererbung kompensiert werden. Leider kennt VB-Script keinen INCLUDE Mechanismus, so dass man diese Klassen von Hand in die Skripte kopieren muss (ein ELO eigener Include Mechanismus ist für kommende ELO Versionen aber bereits in Vorbereitung).





Hinweis: Leider unterscheidet der Befehl GetEntryId(-1) nicht zwischen einem selektierten Archivstartknoten und gar keiner Selektion. In beiden Fällen wird eine 0 zurück gegeben. Wenn Sie die Möglichkeit schaffen wollen, dass auch das ganze Archiv exportiert werden kann, dann müssten Sie den Rückgabewert 0 in 1 übersetzen. Allerdings führt dann auch eine Fehleingabe zum Export des gesamten Archivs. Auf Skript Ebene kann man nicht zwischen den beiden Zuständen unterscheiden.

Dieses Beispiel ist aber immer noch nicht für den produktiven Einsatz geeignet. Es fehlt auch noch eine Fehlerbehandlung und eine Kollisionserkennung. In einem ELO Archiv ist es zulässig, dass in einem Ordner mehrere Dokumente mit gleichem Namen liegen. Im Dateisystem geht das nicht. Das gleiche gilt für Ordner. Deshalb müssen in diesem Fall die Datei- und Ordnernamen bei Bedarf geändert werden. Zudem müssen Fehlerzustände abgefangen werden und dem Anwender gemeldet werden. Hierfür muss die Subroutine ScriptWalk_Process entsprechend erweitert werden:

```
Sub ScriptWalk_Process
  Dim Source, Destination, Ext, Name, Retry

  Const ArchiveStartNode = 9999
  On Error Resume Next
  Exporter.LevelName = PrepareFilePath(Exporter.LevelName)
  call ELO.Status(Exporter.LevelName)

  If (ELO.ObjTypeEx < MaxFolder) Or _
```

```
      (ELO.ObjTypeEx = ArchiveStartNode) Then
  ' Ordner anlegen
  Destination = DestinationBase & Exporter.GetActPath & _
    "\" & Exporter.LevelName

  Retry = 1
  Name = Exporter.LevelName
  While FSO.FolderExists(Destination)
    Exporter.LevelName = Name & "(" & Retry & ")"
    Destination = DestinationBase & Exporter.GetActPath & "\" _
                & Exporter.LevelName & Ext
    Retry = Retry + 1
  Wend
  ELO.DebugOut("Folder: " & Destination)
  ELO.DebugOut(" ")
  Call FSO.CreateFolder(Destination)
  If Err.Number > 0 Then
    call ELO.MsgBox(Err.Number & " " & Err.Description _
          & vbCrLf & Exporter.GetActPath & "\" _
          & Exporter.LevelName, "ELO", vbOkOnly)
  End If
Else
  ' Dokument kopieren
  Source = ELO.GetDocumentPath(ELO.GetEntryId(-2), 0)
  If Source <> "" Then
    Ext = ELO.SplitFileName(Source, 3)
    Destination = DestinationBase & Exporter.GetActPath & _
      "\" & Exporter.LevelName & Ext

    Retry = 1
    Name = Exporter.LevelName
    While FSO.FileExists(Destination)
      Exporter.LevelName = Name & "(" & Retry & ")"
      Destination = DestinationBase & Exporter.GetActPath & "\" _
                  & Exporter.LevelName & Ext
      Retry = Retry + 1
    Wend

    ELO.DebugOut("From: " & Source)
    ELO.DebugOut("To: " & Destination)
    ELO.DebugOut(" ")
    Call FSO.CopyFile(Source, Destination)
    If Err.Number > 0 Then
      call ELO.MsgBox(Err.Number & " " & Err.Description _
            & vbCrLf & Exporter.GetActPath _
```

```
            & "\" & Exporter.LevelName, "ELO", vbOkOnly)
    End If
   End If
  End If
End Sub
```

Refactoring des Exporterskripts

Durch diese Erweiterungen ist die Funktion nun unübersichtlich lang geworden. Zudem passieren hier zu viele unterschiedliche Dinge. Im Allgemeinen sagt man „Eine Funktion sollte nur für eine Aufgabe verantwortlich sein“. Auch wenn das eine recht vage Faustformel ist (was ist denn „eine Aufgabe“?), hat die oben aufgeführte Routine viel Potential für eine übersichtliche Aufteilung. Gerade bei sehr dynamischen Prozessen, wie es die Skript Programmierung nun mal ist, sollten Sie sich hin und wieder die Zeit nehmen und Programmcode, der aus der Form geraten ist, verbessern und überarbeiten.

Als erstes fällt auf, dass an mehreren Stellen ein Dateipfad aus einzelnen Teilen zusammen gesetzt wird. Er beginnt mit einem festen Basispfadanteil, einem relativen Pfad aus der Archivposition, einem Namen und optional einer Dateikennung. Die einzelnen Teile werden aneinander gehangen, lediglich zwischen relativen Pfad und Namen muss zusätzlich ein Backslash eingetragen werden. Wenn sich dieser Aufbau einmal ändert, muss das Skript an vielen Stellen angefasst werden. Wenn dabei nur eine Stelle übersehen wird, gibt es später Fehler in der Ausführung. Als erstes werden wir also eine Funktion einführen, die den Dateipfad zusammensetzt.

```
Function BuildName(Base, Path, Name, Ext)
  BuildName = Base & Path & "\" & Name & Ext
End Function
```

Beim Anlegen eines Ordners oder einer Datei wird im Kollisionsfall ein freier Name gesucht. Auch diese Operation ist ein guter Kandidat für eine mögliche Wiederverwendung, sie wird deshalb in eine eigene Funktion ausgelagert.

```
Function LookupFolderName(Base, ActPath, FolderName)
  Dim Name, Retry, Destination

  Destination = BuildName(Base, ActPath, FolderName, "")

  Retry = 1
  Name = FolderName
  While FSO.FolderExists(Destination)
    FolderName = Name & "(" & Retry & ")"
    Destination = BuildName(Base, ActPath, FolderName, "")
    Retry = Retry + 1
  Wend
  ELO.DebugOut("New Folder: " & Destination)
  ELO.DebugOut("New Name: " & FolderName)

  LookupFolderName = FolderName
End Function
```

Das Anlegen eines Ordners und das Kopieren einer Datei sind auch zwei unterschiedliche Aufgaben, diese sollten aus der Hauptroutine ausgelagert werden.

```
Sub CreateFolder
  Dim Destination, Name, ActPath

  ActPath = Exporter.GetActPath
  Name = LookupFolderName(DestinationBase, _
         ActPath, Exporter.LevelName)
  Destination = BuildName(DestinationBase, ActPath, Name, "")
  Exporter.LevelName = Name

  Call FSO.CreateFolder(Destination)
  If Err.Number > 0 Then
    call ELO.MsgBox(Err.Number & " " & Err.Description _
          & vbCrLf & Destination, "ELO", vbOkOnly)
  End If
End Sub
```

Das gleiche gilt analog für die Ermittlung des Dateinamens und das Kopieren der Datei.

```
Sub CopyFile
```

```
  Dim Source, Destination, Name, Ext, ActPath
  Source = ELO.GetDocumentPath(ELO.GetEntryId(-2), 0)
  If Source <> "" Then
    Ext = ELO.SplitFileName(Source, 3)
    ActPath = Exporter.GetActPath
    Name = LookupFileName(DestinationBase, ActPath, _
          Exporter.LevelName, Ext)
    Destination = BuildName(DestinationBase, ActPath, Name, Ext)
    Exporter.LevelName = Name

    On Error Resume Next
    Call FSO.CopyFile(Source, Destination)
    If Err.Number > 0 Then
      call ELO.MsgBox(Err.Number & " " & Err.Description _
            & vbCrLf & Destination, "ELO", vbOkOnly)
    End If
  End If
End Sub
```

Die eigentliche Hauptroutine wird dadurch wieder übersichtlich und verständlicher.

```
Sub ScriptWalk_Process
  Exporter.LevelName = PrepareFilePath(Exporter.LevelName)
  call ELO.Status(Exporter.LevelName)

  If (ELO.ObjTypeEx < MaxFolder) Or _
      (ELO.ObjTypeEx = ArchiveStartNode) Then
    CreateFolder
  Else
    CopyFile
  End If
End Sub
```

Aus einem 60 Zeilen Monster ist nun eine Gruppe von überschaubaren Einzelfunktionen geworden. Jede einzelne für sich ist leichter zu verstehen. Zudem besteht die Möglichkeit, dass einzelne Funktionen auch an anderen Stellen verwendet werden können.

Das gesamte Modul sieht nun so aus:

```
' Skript: DokumentenExport
' Autor: Matthias Thiele
' Erstellt: 15.06.2010
' Letzte Änderung: 16.06.2010
'
' Dieses Skript läuft mittels der Klasse ScriptWalk
' über alle Nachfolger eines Starteintrags und kopiert
' die Archivstruktur ins Dateisystem.


Option Explicit

Const DestinationBase = "C:\Temp\Export"
Const MaxFolder = 32
Const ArchiveStartNode = 9999

Dim ELO, FSO, StartId, Exporter, Callback
Set ELO = CreateObject("ELO.office")
Set FSO = CreateObject("Scripting.FileSystemObject")

StartId = ELO.GetEntryId(-1)
If StartId = 0 Then StartId = 1
If StartId > 0 Then
  Set Exporter = new ScriptWalk
  Set Callback = GetRef("ScriptWalk_Process")
  Call Exporter.Walk(StartId, MaxFolder, Callback)
  Call ELO.Status("")
End If


Sub ScriptWalk_Process
  Exporter.LevelName = PrepareFilePath(Exporter.LevelName)
  call ELO.Status(Exporter.LevelName)

  If (ELO.ObjTypeEx < MaxFolder) Or _
      (ELO.ObjTypeEx = ArchiveStartNode) Then
    CreateFolder
  Else
    CopyFile
  End If
End Sub


Sub CreateFolder
  Dim Destination, Name, ActPath
```

```
  ActPath = Exporter.GetActPath
  Name = LookupFolderName(DestinationBase, _
         ActPath, Exporter.LevelName)
  Destination = BuildName(DestinationBase, ActPath, Name, "")
  Exporter.LevelName = Name

  On Error Resume Next
  Call FSO.CreateFolder(Destination)
  If Err.Number > 0 Then
    call ELO.MsgBox(Err.Number & " " & Err.Description _
           & vbCrLf & Destination, "ELO", vbOkOnly)
  End If
End Sub


Sub CopyFile
  Dim Source, Destination, Name, Ext, ActPath
  Source = ELO.GetDocumentPath(ELO.GetEntryId(-2), 0)
  If Source <> "" Then
    Ext = ELO.SplitFileName(Source, 3)
    ActPath = Exporter.GetActPath
    Name = LookupFileName(DestinationBase, ActPath, _
           Exporter.LevelName, Ext)
    Destination = BuildName(DestinationBase, ActPath, Name, Ext)
    Exporter.LevelName = Name

    On Error Resume Next
    Call FSO.CopyFile(Source, Destination)
    If Err.Number > 0 Then
      call ELO.MsgBox(Err.Number & " " & Err.Description _
             & vbCrLf & Destination, "ELO", vbOkOnly)
    End If
  End If
End Sub


Function LookupFolderName(Base, ActPath, FolderName)
  Dim Name, Retry, Destination

  Destination = BuildName(Base, ActPath, FolderName, "")

  Retry = 1
  Name = FolderName
  While FSO.FolderExists(Destination)
    FolderName = Name & "(" & Retry & ")"
```

```
    Destination = BuildName(Base, ActPath, FolderName, "")
    Retry = Retry + 1
  Wend
  ELO.DebugOut("New Folder: " & Destination)
  ELO.DebugOut("New Name: " & FolderName)

  LookupFolderName = FolderName
End Function


Function LookupFileName(Base, ActPath, FileName, Ext)
  Dim Name, Retry, Destination

  Destination = BuildName(Base, ActPath, FileName, Ext)

  Retry = 1
  Name = FileName
  While FSO.FileExists(Destination)
    FileName = Name & "(" & Retry & ")"
    Destination = BuildName(Base, ActPath, FileName, Ext)
    Retry = Retry + 1
  Wend
  ELO.DebugOut("New File: " & Destination)
  LookupFileName = FileName
End Function


Function BuildName(Base, Path, Name, Ext)
  BuildName = Base & Path & "\" & Name & Ext
End Function


Function PrepareFilePath(FilePath)
  Dim Reg
  Set Reg = New RegExp

  Reg.Global = True
  Reg.Pattern = "[\*\:\\\&\t\|\r\n<>]"
  PrepareFilePath = Reg.Replace(FilePath, "_")
End Function
```

Dokumentenexport mit Warteanzeige

Mit nur einer Hand voll Änderungen kann man nun zusätzlich noch eine Warteanzeige hinzufügen. Hierzu wird die weiter oben beschriebene Klasse für eine Warteanzeige in das Skript kopiert und die Aufrufe zur Anzeige im Hauptprogramm, sowie die Aufrufe zum Ablauf im der Process Methode hinzugefügt. Änderungen gibt es nur in den ersten 30 Zeilen, der Rest ist identisch zur Vorgängerversion.

```
' Skript: DokumentenExport
' Autor: Matthias Thiele
' Erstellt: 15.06.2010
' Letzte Änderung: 28.06.2010
'
' Dieses Skript läuft mittels der Klasse ScriptWalk
' über alle Nachfolger eines Starteintrags und kopiert
' die Archivstruktur ins Dateisystem.


Option Explicit

Const DestinationBase = "C:\Temp\Export"
Const MaxFolder = 32
Const ArchiveStartNode = 9999

Dim ELO, FSO, StartId, Exporter, Wait, Callback
Set ELO = CreateObject("ELO.office")
Set FSO = CreateObject("Scripting.FileSystemObject")

StartId = ELO.GetEntryId(-1)
If StartId = 0 Then StartId = 1
If StartId > 0 Then
  Set Wait = New ExplorerWaitBar
  Call Wait.Show("Dokumentenexport", "Starte Export Vorgang")
  Set Callback = GetRef("ScriptWalk_Process")
  Set Exporter = new ScriptWalk
  Call Exporter.Walk(StartId, MaxFolder, Callback)
  Call ELO.Status("")
  Call Wait.Hide("Export abgeschlossen", 1000)
End If


Sub ScriptWalk_Process
```

```
  Exporter.LevelName = PrepareFilePath(Exporter.LevelName)
  Call ELO.Status(Exporter.LevelName)
  Call Wait.Update(Exporter.LevelName)

  If (ELO.ObjTypeEx < MaxFolder) Or _
     (ELO.ObjTypeEx = ArchiveStartNode) Then
    CreateFolder
  Else
    CopyFile
  End If
End Sub


Sub CreateFolder
  Dim Destination, Name, ActPath
  ActPath = Exporter.GetActPath
  Name = LookupFolderName(DestinationBase, _
         ActPath, Exporter.LevelName)
  Destination = BuildName(DestinationBase, ActPath, Name, "")
  Exporter.LevelName = Name

  On Error Resume Next
  Call FSO.CreateFolder(Destination)
  If Err.Number > 0 Then
    call ELO.MsgBox(Err.Number & " " & Err.Description _
          & vbCrLf & Destination, "ELO", vbOkOnly)
  End If
End Sub


Sub CopyFile
  Dim Source, Destination, Name, Ext, ActPath
  Source = ELO.GetDocumentPath(ELO.GetEntryId(-2), 0)
  If Source <> "" Then
    Ext = ELO.SplitFileName(Source, 3)
    ActPath = Exporter.GetActPath
    Name = LookupFileName(DestinationBase, ActPath, _
           Exporter.LevelName, Ext)
    Destination = BuildName(DestinationBase, ActPath, Name, Ext)
    Exporter.LevelName = Name

    On Error Resume Next
    Call FSO.CopyFile(Source, Destination)
    If Err.Number > 0 Then
      call ELO.MsgBox(Err.Number & " " & Err.Description _
```

```
            & vbCrLf & Destination, "ELO", vbOkOnly)
    End If
  End If
End Sub


Function LookupFolderName(Base, ActPath, FolderName)
  Dim Name, Retry, Destination

  Destination = BuildName(Base, ActPath, FolderName, "")

  Retry = 1
  Name = FolderName
  While FSO.FolderExists(Destination)
    FolderName = Name & "(" & Retry & ")"
    Destination = BuildName(Base, ActPath, FolderName, "")
    Retry = Retry + 1
  Wend
  ELO.DebugOut("New Folder: " & Destination)
  ELO.DebugOut("New Name: " & FolderName)

  LookupFolderName = FolderName
End Function


Function LookupFileName(Base, ActPath, FileName, Ext)
  Dim Name, Retry, Destination

  Destination = BuildName(Base, ActPath, FileName, Ext)

  Retry = 1
  Name = FileName
  While FSO.FileExists(Destination)
    FileName = Name & "(" & Retry & ")"
    Destination = BuildName(Base, ActPath, FileName, Ext)
    Retry = Retry + 1
  Wend
  ELO.DebugOut("New File: " & Destination)
  LookupFileName = FileName
End Function


Function BuildName(Base, Path, Name, Ext)
  BuildName = Base & Path & "\" & Name & Ext
End Function
```

```
Function PrepareFilePath(FilePath)
  Dim Reg
  Set Reg = New RegExp

  Reg.Global = True
  Reg.Pattern = "[\*\:\\\&\t\|\r\n<>]"
  PrepareFilePath = Reg.Replace(FilePath, "_")
End Function


Class ScriptWalk
  Private Level
  Private LocalMaxLevel
  Private ActPath
  Private ProcessFunction
  Private M_LevelName

  Public Sub Walk(StartId, MaxLevel, Callback)
    Level = 0
    ActPath = ""
    LocalMaxLevel = MaxLevel
    Set ProcessFunction = Callback

    Call LocalWalk(StartId)
  End Sub

  Public Function GetActPath
    GetActPath = ActPath
  End Function

  Public Property Get LevelName
    LevelName = M_LevelName
  End Property

  Public Property Let LevelName(NewName)
    M_LevelName = NewName
  End Property

  Private Sub LocalWalk(StartId)
    Dim ChildListString, ChildList, i, Msg, OldPath

    ' Zuerst den Eintrag selber einlesen und prüfen
    If ELO.PrepareObjectEx(StartId, 0, 0) < 0 Then
```

```
      ' Der Eintrag konnte nicht gelesen werden, hier abbrechen
      Exit Sub
    End If

    M_LevelName = ELO.ObjShort
    Call ProcessFunction

    ' Nun die Nachfolgerliste prüfen. Nur Ordner haben Nachfolger
    If ((ELO.ObjTypeEx <= MaxFolder) Or _
        (ELO.ObjTypeEx = 9999)) and _
        (Level < LocalMaxLevel) Then
      ChildListString = ELO.CollectChildList(StartId)
      If ChildListString <> "" Then
        ChildList = Split(ChildListString, ":")
        Level = Level + 1
        OldPath = ActPath
        ActPath = ActPath & "\" & M_LevelName
        For i = 0 to UBound(ChildList)
          If ChildList(i) <> "" Then
            Call LocalWalk(ChildList(i))
          End If
        Next
        Level = Level - 1
        ActPath = OldPath
      End If
    End If
  End Sub
End Class


' Hier beginnt der Funktionsbereich für die Klasse der Statusanzeige
' Die Statusanzeige wird durch Methodenaufrufe in folgender
Reihenfolge
' verwendet
'
'   Show
'   mehrfaches Update
'   Hide

Class ExplorerWaitBar

Private Explorer

' Zeigt die Statusanzeige an
Public Sub Show(Titel, StartMessage)
```

```
  Set Explorer = CreateObject("InternetExplorer.Application")
  Explorer.Navigate "about:blank"
  Explorer.ToolBar = 0
  Explorer.StatusBar = 0
  Explorer.Left = 400
  Explorer.Top = 300
  Explorer.Width=370
  Explorer.Height = 160


  Do While (Explorer.Busy)
    call ELO.Sleep(0, 200)
  Loop

  Explorer.Visible = 1
  call PrepareHtml(Titel, StartMessage)
End Sub



' Aktualisiert die Warteanzeige
Public Sub Update(Message)
  Dim Doc, Div

  Set Doc = Explorer.Document
  Set Div = Doc.getElementById("msg")
  Div.innerHTML = Message
End Sub


' Schließt die Statusanzeige wieder
Public Sub Hide(ReadyMessage, ShowDelay)
  Explorer.Document.Body.InnerHTML = ReadyMessage
  call ELO.Sleep(0, ShowDelay)
  Explorer.Quit
End Sub

Private Sub PrepareHtml(Title, StartMessage)
  Dim Hdr, Body, Start, Doc

  Hdr ="<head><title>" & Title & "</title>" & _
       "<script type=""text/javascript"">" & _
       "var leftpos = 0;" & _
       "var innerpos = 0;" & _
       "var direction = 1;" & _
```

```
    "function tick() {" & _
    " var outer = document.getElementById(""outer"");" & _
    " var inner = document.getElementById(""inner"");" & _
    " if (inner && outer) {" & _
    " if (direction == 1) {" & _
    "   if (innerpos < 72) {" & _
    "     innerpos++;" & _
    "     inner.style.left = innerpos + ""px"";" & _
    "   } else {" & _
    "     leftpos++;" & _
    "     outer.style.left = leftpos + ""px"";" & _
    "     if (leftpos > 120) {" & _
    "       direction = -1;" & _
    "     }" & _
    "   }" & _
    " } else {" & _
    "   if (innerpos > 0) {" & _
    "     innerpos--;" & _
    "     inner.style.left = innerpos + ""px"";" & _
    "   } else {" & _
    "     leftpos--;" & _
    "     outer.style.left = leftpos + ""px"";" & _
    "     if (leftpos < 1) {" & _
    "       direction = 1;" & _
    "     }" & _
    "   }" & _
    " }" & _
    "}" & _
    "}" & _
    "</script></head>"

Body = "<body scroll=""no"" onload=""setInterval('tick()', 20);"">" & _
      "<div style=""border:solid 1px;width:203px;color:#808080"">" & _
      "<div id=""outer"" style=""position:relative;left:0px;" & _
      "width:80px;background-color:#f0f0fa;height:30px"">" & _
      "<div id=""inner"" style=""position:relative;left:0px;width:8px;" & _
      "background-color:#00ff80;height:30px"">" & _
      "<div style=""position:relative;left:0px;width:4px;" & _
      "background-color:#ffc840;height:30px"">" & _
      " </div></div></div></div><div id=""msg"">" & _
      StartMessage & "</div></body>"

Start = "<html>" & Hdr & Body & "</html>"
Set Doc = Explorer.Document
Doc.Open
```

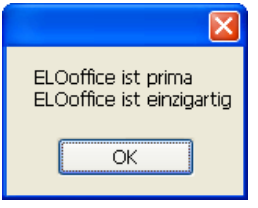
```
  Call Doc.Write(Start)
  Doc.Close
End Sub

End Class
```

Reguläre Ausdrücke in VB-Script

In dem Kapitel über den Dokumentenexport ist schon einmal kurz der Begriff „Regulärer Ausdruck“ aufgeblitzt. In diesem Kapitel soll nun die Funktionsweise und Verwendung von regulären Ausdrücken behandelt werden.

Ein regulärer Ausdruck ist eine formale Beschreibung zur Erkennung bestimmter Zeichenfolgen. Stark vereinfacht könnte man auch die Instr oder Replace Funktion als eine Verwendung von eingeschränkten regulären Ausdrücken ansehen. In diesen Fällen wird nach einem festen Text gesucht. In der Replace Funktion wird ein Text (1. Parameter „InputString“ im Inhalt nach einem Teiltext (2. Parameter „prima“) durchsucht und jedes Vorkommen durch einen anderen Text (3. Parameter „einzigartig“) ersetzt.

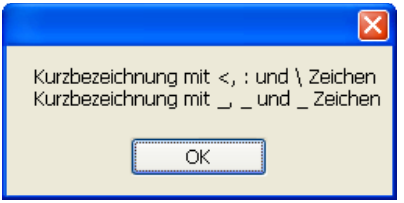
```
InputString = "ELOoffice ist prima"
OutputString = Replace(InputString, "prima", "einzigartig")
MsgBox InputString & vbCrLf & OutputString
```



Suchen und Ersetzen mit regulären Ausdrücken

In vielen Fällen ist die Suche nach einem festen Text ausreichend, manchmal aber auch nicht. Es ist zum Beispiel denkbar, dass nach mehreren unterschiedlichen Texten gesucht werden soll. Wenn eine Kurzbezeichnung in einen Dateinamen umgesetzt werden soll, dann müssen alle Doppelpunkte, < -Zeichen und Backslashs in ein anderes, harmloses Zeichen umgewandelt werden, z.B. einen Unterstrich. Mittels der Replace Funktion sieht das dann so aus:

```
InputString = "Kurzbezeichnung mit <, : und \ Zeichen"
OutputString = Replace(InputString, "<", "_")
OutputString = Replace(OutputString, ":", "_")
OutputString = Replace(OutputString, "\", "_")
MsgBox InputString & vbCrLf & OutputString
```



Tatsächlich sind nicht nur diese drei Zeichen kritisch, es gibt noch eine Reihe mehr. Dann umfasst die Replace Kette schnell zehn oder zwanzig Zeilen. Das ist nicht nur langsam in der Ausführung, es ist auch unübersichtlich und schlecht wartbar. Hier kann nun ein Replace mit einem regulären Ausdruck verwendet werden. Gewünscht wird so einen Funktion:

```
' Diese Funktion gibt es nicht
OutputString = Replace(InputString, "<" OR ":" OR "\" OR "_")
```

Hier kommt nun ein regulärer Ausdruck ins Spiel, dort kann man so etwas formulieren:

```
InputString = "Kurzbezeichnung mit <, : und \ Zeichen"
Set Reg = New RegExp
Reg.Global = True
Reg.Pattern = "[<:\\]"
OutputString = Reg.Replace(InputString, "_")
MsgBox OutputString
```

Zur Verwendung wird zuerst ein Objekt für den regulären Ausdruck erzeugt (New RegExp). Das Property Global bestimmt, ob nur der erste Treffer oder alle Treffer bearbeitet werden. Da wir alle kritischen Zeichen aus der Kurzbezeichnung ausfiltern wollen, muss es auf True gesetzt werden.

Als nächstes kommt der eigentliche reguläre Ausdruck: [<:\\] – das sieht auf den ersten Blick unverständlich aus. Tatsächlich sind reguläre Ausdrücke schwer zu lesen. Aber wenn man die wichtigsten Regeln kennt, kann man sie relativ leicht zusammensetzen.

Zum Verständnis des Beispiels muss man folgendes wissen:

1. Wenn mehrere unterschiedliche Zeichen als mögliche Treffer zur Wahl stehen, dann werden sie in eine eckige Klammer gesetzt. A oder B oder C wird dann so formuliert: [ABC].
2. Damit man auch nach eckigen Klammern und anderen Sonderzeichen suchen kann, werden sie durch einen Backslash markiert: \[steht für eine öffnende eckige Klammer, \] für die schließende Klammer. Wenn man nun nach einer öffnenden oder schließenden Klammer sucht, dann wird das so formuliert: [\[\]]. Ein Backslash wird als \\ ausgegeben, ein Stern als *. Eine vollständige Liste aller Sonderzeichen finden Sie hier: http://msdn.microsoft.com/en-us/library/ms974570.aspx

Mit diesem Wissen kann man nun den Ausdruck analysieren: [<:\\]

[<:\\] Rot: die eckige Klammer markiert eine ODER Auswahl

Gelb: entweder das Zeichen „kleiner“ <

Grün: oder den Doppelpunkt :

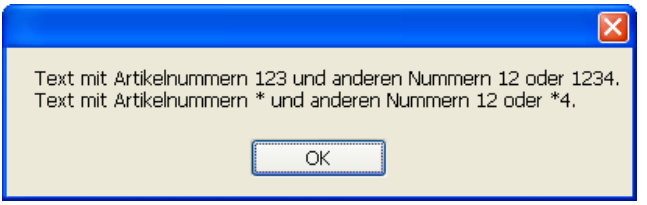
Blau: oder einen Backslash \ (als Escape \\ geschrieben)

Es kann aber nicht nur nach einfachen Zeichen oder Texten gesucht werden. Wenn man z.B. nach einer 3-stelligen Artikelnummer in einem Text sucht, dann könnte man nach 001 oder 002 oder 003 oder … oder 997 oder 998 oder 999 suchen. Das ist offensichtlich nicht praktikabel.

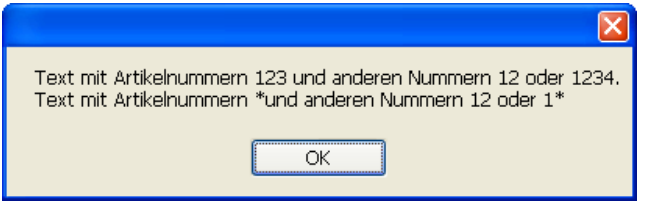
Wie man nach einer Ziffer sucht, das ist schon aus dem Beispiel zuvor bekannt: [0123456789]. Diese Schreibweise kann man zum Glück auch abkürzen: [0-9]. Nun bestehen diese Artikelnummern aber aus 3 Ziffern

direkt hintereinander. Wie der Ausdruck dafür aussieht kann man vielleicht erraten: [0-9][0-9][0-9]. Auch hier gibt es wieder eine vereinfachte Schreibweise: [0-9]{3}.

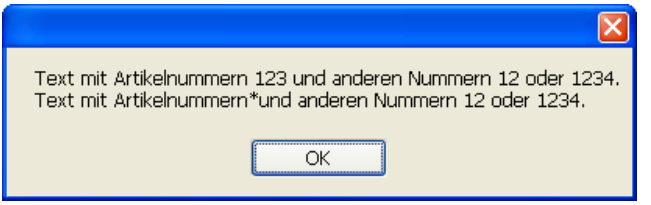
```
InputString = "Text mit Nummern 123 und Nummern 12 oder 1234."
Set Reg = New RegExp
Reg.Global = True
Reg.Pattern = "[0-9]{3}"
OutputString = Reg.Replace(InputString, "*")
MsgBox InputString & vbCrLf & OutputString
```



Im Beispiel sieht man nun, dass die dreistellige Nummer durch ein * ersetzt wurde und die zweistellige nicht. Wir haben unser Ziel also fast erreicht, aber noch nicht ganz, denn die vierstellige Zahl wurde auch ersetzt, zumindest die ersten 3 Ziffern davon. Denn eine vierstellige Zahl ist eine dreistellige Zahl mit einer zusätzlichen Ziffer am Ende. Wenn wir nur dreistellige Zahlen ersetzen wollen, dann müssen wir in dem Ausdruck mitteilen, dass nach der dritten Ziffer etwas anderes als eine weitere Ziffer kommen muss: [0-9]{3}[^0-9]. Das ^-Zeichen in der letzten Gruppe steht für „Nicht vorhanden“.

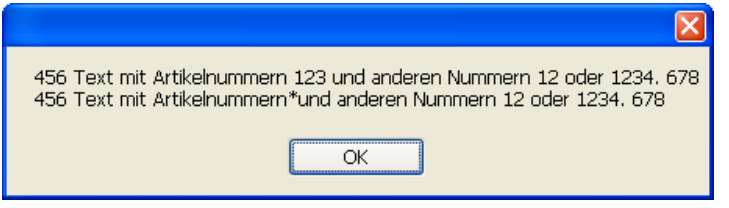


Hoppla, das war es doch nicht. Denn eine vierstellige Zahl ist auch eine dreistellige Zahl mit einer führenden Ziffer. Also eine neuer Versuch mit: [^0-9][0-9]{3}[^0-9]



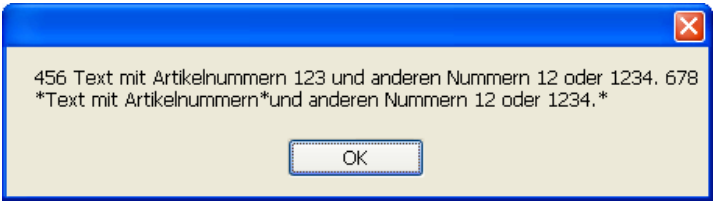
Hinweis: Statt [0-9] kann man kürzer auch \d schreiben. Es ist aber weniger leicht zu lesen und auch kaum kürzer. Welche Variante man einsetzt, ist eine persönliche Entscheidung: [^\d]\d{3}[^\d].

Die Musterlösung hat jetzt noch den Fehler, dass eine Artikelnummer direkt am Anfang oder direkt am Ende des Textes nicht erkannt werden würde. Wenn dieser Fall vorkommen kann, muss man das auch noch im regulären Ausdruck berücksichtigen.



Für „Ganz am Anfang“ und „Ganz am Ende gibt es auch vordefinierte Sonderzeichen ^ bzw. $. Am besten verdeutlicht man sich noch mal, was man erreichen möchte: (Textanfang ODER nicht Ziffer) danach 3 Ziffern danach (nicht Ziffer ODER Textende). Das ODER drückt man in regulären Ausdrücken mit dem Pipe Symbol | (kein kleines L) aus, „Textanfang ODER nicht Ziffer“ sieht dann so aus: ^|[^0-9]. Analog dazu „nicht Ziffer ODER Textende“: [^0-9]|$. In der Gruppierung müssen diese dann jeweils geklammert werden, damit die Zuordnung stimmt und der Gesamtausdruck sieht nun so aus: (^|[^0-9])[0-9]{3}([^0-9]|$)

Hinweis: Das ^-Zeichen kann je nach Position für „Nicht vorhanden“ oder Textanfang stehen. Leider sind reguläre Ausdrücke nur schwer lesbar, man muss immer sehr genau hinsehen, was passiert. Deshalb ist es sinnvoll, diese zusätzlich in einem Kommentar im Programm zu beschreiben.




```
InputString = "456 Text mit Nummern 123 und 12 oder 1234. 678"
Set Reg = New RegExp
Reg.Global = True
Reg.Pattern = "(^|[^0-9])[0-9]{3}([^0-9]|$)"
OutputString = Reg.Replace(InputString, "*")
MsgBox InputString & vbCrLf & OutputString
```

Das Beispiel zeigt zweierlei:

1. mit einem regulären Ausdruck kann man viel erreichen…
2. aber es ist nicht ganz einfach.

Zur Konstruktion eines Ausdrucks muss man sich vorher gut überlegen, was man erreichen will. Am besten schreibt man sich dann den Ausdruck in normaler Sprache auf. Diese Darstellung kann man dann in die formale Sprache der regulären Ausdrücke übersetzen. Zum Testen sollte man sich dann Texte mit den normalen Inhalten und möglichst allen Sonderfällen erstellen und durch den regulären Ausdruck auswerten lassen.

Trefferliste mit regulären Ausdrücken

Das nächste Beispiel geht davon aus, dass aus einem Text alle Beträge gefunden werden sollen.

In kaufmännischen Belegen gibt es alle möglichen und unmöglichen Notationen für Beträge, wegen der Übersichtlichkeit gehen wir aber davon aus, dass ein Betrag optional mit einem Plus oder Minus beginnt, danach folgen beliebig viele Ziffern, dann kommt ein Komma und noch mal zwei Ziffern.

Optional + oder – (evtl. auch gar nichts)	(\+|-)*
Beliebig viele Ziffern (mindestens eine)	[0-9]+
Komma	,
Zwei Ziffern für die Nachkommastellen	[0-9][0-9]
Danach aber keine weiteren Ziffern (1,234 ist kein Betrag)	([^0-9]|$)

Der reguläre Ausdruck sieht dann so aus: (\+|-)*[0-9]+,[0-9][0-9]([^0-9]|$)

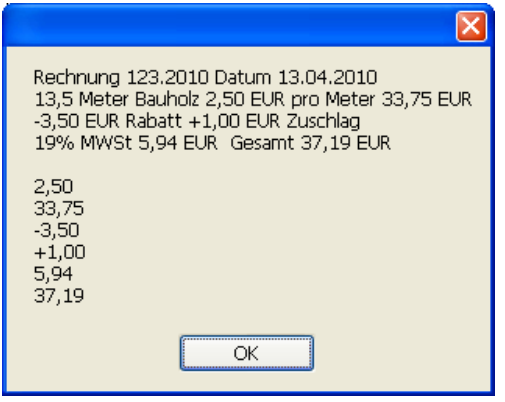
In diesem Beispiel wollen wir die Beträge nicht einfach aus dem Text löschen, wir wollen Sie für eine Weiterverarbeitung als Liste zurück bekommen. Deshalb wird nun nicht mit Replace gearbeitet sondern die Methode Execute aufgerufen. Diese Methode liefert eine Liste aller Treffer zurück:

```
Reg.Pattern = "(\+|-)*[0-9]+,[0-9][0-9]([^0-9]|$)"
Set Matches = Reg.Execute(InputString)
```

Beachten Sie, dass hier keine einfache Zuweisung erfolgt, da Execute keinen einfachen Datentyp (String, Zahl…) sondern ein Objekt zurück gibt. In diesen Fällen muss immer mit Set gearbeitet werden. Ein Skript, welches die Beträge herauszieht und mittels MsgBox ausgibt sieht dann so aus:

```
InputString = "Rechnung 123.2010 Datum 13.04.2010 " & vbCrLf  _
        & "13,5 Meter Bauholz 2,50 EUR pro Meter 33,75 EUR " & vbCrLf _
        & "-3,50 EUR Rabatt +1,00 EUR Zuschlag " & vbCrLf _
        & "19% MWSt 5,94 EUR  Gesamt 37,19 EUR" & vbCrLfSet Reg =
New RegExp
Reg.Global = True
```

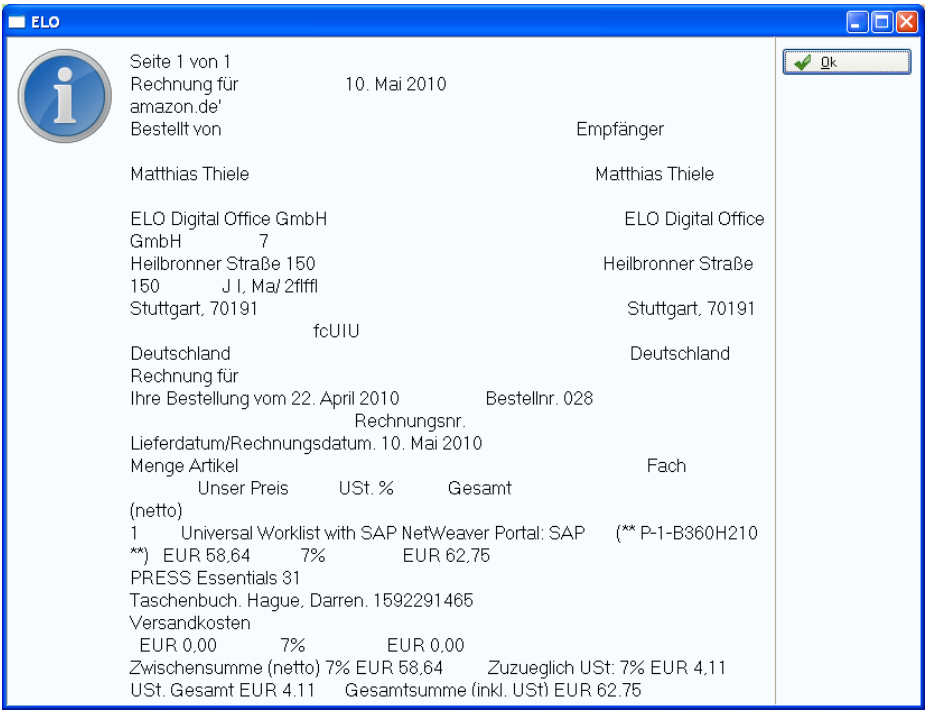
```
Reg.Pattern = "(\+|-)*[0-9]+,[0-9][0-9]([^0-9]|$)"
Set Matches = Reg.Execute(InputString)
Msg = ""
For Each Match in Matches
  Msg = Msg & Match & vbCrLf
Next
MsgBox InputString & vbCrLf & Msg
```



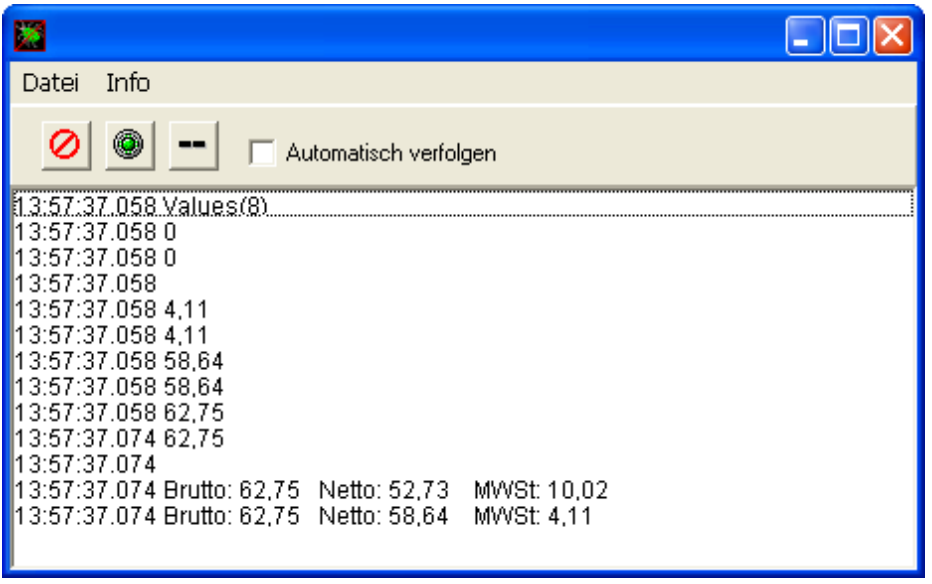
Rechnungsbeträge erkennen

Dieses Skript kann man nun zu einer einfachen Erkennung der Rechnungsbeträge ausbauen. Zur automatischen Erkennung wird davon ausgegangen, dass auf der Rechnung sowohl der Brutto-, wie auch der Nettobetrag und die Mehrwertsteuer ausgewiesen werden. Zur Analyse werden alle Beträge mit einem MWSt Satz von 19% und 7% darauf hin untersucht, ob es eine passende Betragskombination gibt. Der größte Treffer wird als Rechnungsbetrag angenommen. Diese einfache Heuristik funktioniert natürlich nicht, wenn die Mehrwertsteuer einen abweichenden Wert besitzt (z.B. 0%) oder die Rechnung aus gemischten Mehrwertsteuersätzen zusammengesetzt ist.

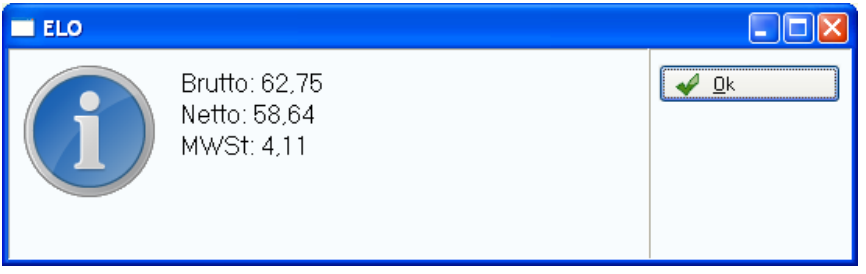
Das Skript führt zuerst eine OCR Analyse über die komplette Seite durch.



Aus dem Text werden dann alle Beträge herausgezogen und auf passende Kombinationen hin untersucht.



Der größte Treffer wird dann als Ergebnis ausgewiesen.



Obwohl das Skript relativ lang ist, bleibt es durch die klare Reihenfolge der Abläufe sehr übersichtlich.

Das Skript ermittelt zuerst die Dokumentendatei zum aktuell ausgewählten Dokument und übergibt den Namen an die Analyse Funktion. Diese ist in eine eigene Klasse eingepackt um eine leichtere Wiederverwendbarkeit auch in anderen Projekten zu ermöglichen.

Id = ELO.GetEntryId(-1)

```
If Id > 0 Then
  Path = ELO.GetDocumentPath(Id, 0)
  If Path <> "" Then
    Set Find = new FindAmount
    Call ELO.Status("OCR: Rechnungstext einlesen")

    if Find.ProcessDocument(Path) Then
      Call ELO.MsgBox("Brutto: " & Find.Brutto & vbCrLf & _
                "Netto: " & Find.Netto & vbCrLf & _
                "MWSt: " & Find.Mwst, "ELO", vbOkOnly)
    Else
      Call ELO.MsgBox("Es wurde keine geeignete " _
                 & "Kombination von Beträgen gefunden", _
                 "ELO", vbOkOnly)
    End If

    Call ELO.Status("")
  End If
End If
```

Zur Analyse der Datei werden die ELO Formularerkennungsfunktionen eingesetzt. Es wird zuerst die Liste der OCR Rechtecke gelöscht und ein neues Rechteck über das gesamte Dokument aufgespannt (von 0, 0 bis 999, 999 Promille). Anschließend wird die OCR Analyse aufgerufen und das Ergebnis in die Funktion ProcessText übergeben.

Falls man sich einen Überblick über die Qualität des OCR Ergebnisses verschaffen will, kann man das Kommentarzeichen von der ELO.MsgBox entfernen und sich so den Text anzeigen lassen.

```
  Public Function ProcessDocument(Path)
    Call ELO.OcrClearRect
    Call ELO.OcrAddRect("0, 0, 999, 999")
    Call ELO.OcrAnalyze(Path, 0)
    'Call ELO.MsgBox(ELO.OcrGetText(0), "ELO", vbOkOnly)
    Call ProcessText(ELO.OcrGetText(0))
    ProcessDocument = Brutto <> 0.0
  End Function
```

Innerhalb von ProcessText werden die Beträge in der Funktion MatchValues mittels des weiter oben aufgeführten regulären Ausdrucks ausgelesen und in sortierter Reihenfolge zurück gegeben. Es wird dann vom größten Betrag an abwärts nach einer passenden Brutto – Netto – Mehrwertsteuer Kombination gesucht und beim ersten Treffer abgebrochen.

```
Private Sub ProcessText(Text)
  Dim Sorted, i

  Sorted = MatchValues(Text)
  Call ShowValues(Sorted)

  For i = Ubound(Sorted) to 1 step -1
    if CalcAndCheck(i, MwstNormal, Sorted) Then
      Exit For
    end if

    if CalcAndCheck(i, MwstErmaessigt, Sorted) Then
      Exit For
    end if

    M_Brutto = 0.0
    M_Netto = 0.0
    M_Mwst = 0.0
  Next

End Sub
```

Der reguläre Ausdruck erkennt alle Zeichenfolgen mit einem optionalen + oder – am Anfang, gefolgt von beliebig vielen Ziffern, einem Komma und nochmals zwei Ziffern, als Beträge. Sie werden in einem Array gesammelt und sortiert zurück gegeben. Beim Auswerten der Beträge ist zu beachten, dass über den Teilausdruck „([^0-9]|$)“ am Ende ein weiteres Zeichen (aber keine Ziffer!) auftreten kann. Dieses muss dann bei Bedarf entfernt werden.

```
Private Function MatchValues(Text)
  Dim Reg, Matches, Match, Values(), i , Val, Last
  Set Reg = New RegExp
  Reg.Global = True
  Reg.Pattern = "(\+|-)*[0-9]+,[0-9][0-9]([^0-9]|$)"
```

```
  Set Matches = Reg.Execute(Text)
  Redim Values(Matches.Count)
  For i = 0 To Matches.Count - 1
    Val = Matches(i).Value
    Last = Right(Val, 1)
    If (Last < "0") Or (Last > "9") Then
      Val = Left(Val, Len(Val) - 1)
    End If

    Values(i) = CDbl(Val)
  Next
  MatchValues = BubbleSort(Values)
End Function
```

Verwendung von SubMatches

Wenig schön ist die Behandlung des zusätzlichen Zeichens am Ende der Ausgabe. Zum Glück kann man es über einen passend formatierten regulären Ausdruck besser machen. Jeder Treffer eines Regulären Ausdrucks besteht aus Teilausdrücken, die jeweils in runde Klammern eingefasst werden. Diese Teilausdrücke können auch einzeln angesprochen werden. In der aktuellen Version sind schon der Vorzeichenbereich und das Zusatzzeichen am Ende als Bereich markiert, es fehlt nur noch ein eigener Bereich für die eigentliche Zahl.

```
Reg.Pattern = "(\+|-)*([0-9]+,[0-9][0-9])([^0-9]|$)"
```

Nun besteht jeder Treffer aus drei Teilen: dem Vorzeichen, der Zahl und dem Folgezeichen. Das Vorzeichen und das Folgezeichen können leer sein, aber auch dann ist der Bereich vorhanden, als Leerstring. Beim Einlesen der Werte interessiert nur das Vorzeichen (Bereich 0) und die Zahl (Bereich 1). Das Folgezeichen (Bereich 2) wird ignoriert. Auf die Teilbereiche kann mittels der Funktion SubMatches() zugegriffen werden, als Parameter wird die Nummer des Teilbereichs angegeben. Die Schleife zum Einlesen aller Zahlen sieht dann so aus:

```
Reg.Pattern = "(\+|-)*([0-9]+,[0-9][0-9])([^0-9]|$)"
```

```
Set Matches = Reg.Execute(Text)
Redim Values(Matches.Count)
For i = 0 To Matches.Count - 1
  Set Val = Matches(i)
  Values(i) = CDbl(Val.SubMatches(0) & Val.SubMatches(1))
Next
```

> Hinweis: Für eine echte Anwendung sollte man sich hier noch Gedanken über weitere Zahlenformate machen, z.B. Tausendertrennung (1.200,30) und unterschiedliche Nachkommaabtrennung (1200,30 oder 1200.30).

In der Funktion CalcAndCheck wird zuerst zu jedem Betrag, der als Brutto in Frage kommen könnte, der dazu passende Netto Betrag und die Mehrwertsteuer ausgerechnet. Über die Unterfunktion ValueExists wird dann geprüft, ob diese Beträge ebenfalls in der Trefferliste vorkommen.

```
Private Function CalcAndCheck(Index, MwstProzent, ByRef Values)
  M_Brutto = Values(Index)
  M_Netto = Round(M_Brutto / ((100.0 + MwstProzent)/100.0), 2)
  M_Mwst = M_Brutto - M_Netto
  Call ELO.DebugOut("Brutto: " & M_Brutto & "   Netto: " & _
    M_Netto & "    MWSt: " & M_Mwst)
  CalcAndCheck = ValueExists(M_Netto, Values) And _
    ValueExists(M_Mwst, Values)
End Function
```

Das komplette Skript sieht dann so aus:

```
' Skript: Rechnungsbeträge auffinden
' Autor: Matthias Thiele
' Erstellt: 16.06.2010
' Letzte Änderung: 18.06.2010
'
' Dieses Programm führt auf dem aktuell selektierten
' Dokument einen OCR Vorgang über das gesamte Bild
' aus und ermittelt alle Beträge aus dem gefundenen
' Text. Daraus wird dann nach Möglichkeit der
' Rechnungsbetrag ermittelt
```

```
Option Explicit


Dim ELO, Id, Path, Find
Set ELO = CreateObject("ELO.office")

Id = ELO.GetEntryId(-1)
If Id > 0 Then
  Path = ELO.GetDocumentPath(Id, 0)
  If Path <> "" Then
    Set Find = new FindAmount
    Call ELO.Status("OCR: Rechnungstext einlesen")

    if Find.ProcessDocument(Path) Then
      Call ELO.MsgBox("Brutto: " & Find.Brutto & vbCrLf & _
                "Netto: " & Find.Netto & vbCrLf & _
                "MWSt: " & Find.Mwst, "ELO", vbOkOnly)
    Else
      Call ELO.MsgBox("Es wurde keine geeignete " _
                 & "Kombination von Beträgen gefunden", _
                 "ELO", vbOkOnly)
    End If

    Call ELO.Status("")
  End If
End If


' Die Klasse FindAmount hat eine öffentliche Funktion
' ProcessDocument, diese erhält als Parameter einen
' Dateinamen und untersucht daraufhin das Dokument
' auf zueinander passende Beträge für Brutto, Netto
' und Mwst. Die größte ermittelte Kombination wird
' als Ergebnis in den Properties Brutto, Netto, Mwst
' gespeichert.

Const MwstNormal = 19
Const MwstErmaessigt = 7


Class FindAmount
  Private M_Brutto, M_Netto, M_Mwst
```

```
Public Property Get Brutto
  Brutto = M_Brutto
End Property

Public Property Get Netto
  Netto = M_Netto
End Property

Public Property Get Mwst
  Mwst = M_Mwst
End Property

Public Function ProcessDocument(Path)
  Call ELO.OcrClearRect
  Call ELO.OcrAddRect("0, 0, 999, 999")
  Call ELO.OcrAnalyze(Path, 0)
  'Call ELO.MsgBox(ELO.OcrGetText(0), "ELO", vbOkOnly)
  Call ProcessText(ELO.OcrGetText(0))
  ProcessDocument = Brutto <> 0.0
End Function


Private Sub ProcessText(Text)
  Dim Sorted, i

  Sorted = MatchValues(Text)
  Call ShowValues(Sorted)

  For i = Ubound(Sorted) to 1 step -1
    if CalcAndCheck(i, MwstNormal, Sorted) Then
      Exit For
    end if

    if CalcAndCheck(i, MwstErmaessigt, Sorted) Then
      Exit For
    end if

    M_Brutto = 0.0
    M_Netto = 0.0
    M_Mwst = 0.0
  Next

End Sub
```

```
Private Function MatchValues(Text)
  Dim Reg, Matches, Match, Values(), i , Val, Last
  Set Reg = New RegExp
  Reg.Global = True
  Reg.Pattern = "(\+|-)*([0-9]+,[0-9][0-9])([^0-9]|$)"
  Set Matches = Reg.Execute(Text)
  Redim Values(Matches.Count)
  For i = 0 To Matches.Count - 1
    Set Val = Matches(i)
    Values(i) = CDbl(Val.SubMatches(0) & Val.SubMatches(1))
  Next
  MatchValues = BubbleSort(Values)
End Function


Private Function CalcAndCheck(Index, MwstProzent, ByRef Values)
  M_Brutto = Values(Index)
  M_Netto = Round(M_Brutto / ((100.0 + MwstProzent)/100.0), 2)
  M_Mwst = M_Brutto - M_Netto
  Call ELO.DebugOut("Brutto: " & M_Brutto & "   Netto: " & _
    M_Netto & "    MWSt: " & M_Mwst)
  CalcAndCheck = ValueExists(M_Netto, Values) And _
    ValueExists(M_Mwst, Values)
End Function


Private Function ValueExists(V, ByRef Values)
  Dim i
  ValueExists = FALSE
  For i = 0 To Ubound(Values)
    If Abs(Values(i) - V) < 0.011 Then
      ValueExists = TRUE
      Exit For
    End If
  Next
End Function


Private Function Bubblesort(ArrSort)
  Dim i, j, ArrTemp
  For i = 0 to Ubound(ArrSort)
    For j = i + 1 to Ubound(ArrSort)
      If ArrSort(i) > ArrSort(j) Then
        ArrTemp = ArrSort(i)
        ArrSort(i) = ArrSort(j)
```

```
        ArrSort(j) = ArrTemp
      End If
    Next
  Next

  Bubblesort = ArrSort
End Function


Private Sub ShowValues(ByRef Values)
  Dim i
  Call ELO.DebugOut("Values(" & Ubound(Values) & ")")
  For i = 0 to Ubound(Values)
    Call ELO.DebugOut(Values(i))
  Next

  Call ELO.DebugOut("")
End Sub

End Class
```

Fehlerbehandlung in VB-Script

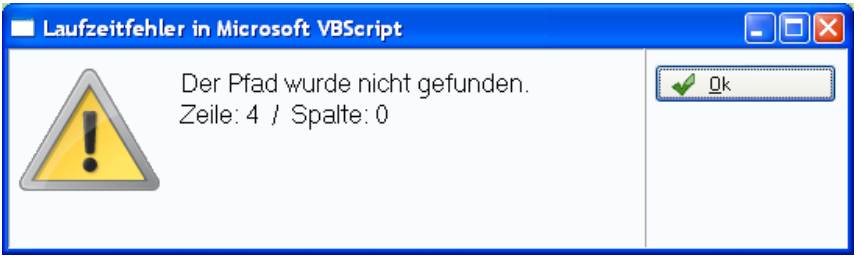
Die Fehlerbehandlung in Skripten ist oft ein kritisches Thema. In Testumgebungen wird mit definierten Daten in bekannten Umgebungen gearbeitet. Wenn es dann aber in den laufenden Betrieb geht, dann treten mitunter ganz andere Bedingungen auf. Wenn es nun keine gute Fehlerbehandlung gibt, dann bricht das Skript unerwartet ab. Oder noch schlimmer: es arbeitet scheinbar normal weiter, produziert aber falsche Ergebnisse.

Leider ist die Unterstützung im Fehlerfall unter VB-Script nur sehr schwach ausgeprägt. Zudem sind die Konsequenzen nicht immer offensichtlich. Deshalb werden im Folgenden ein paar Fehlersituationen untersucht.

Fehler im Hauptprogramm

Wenn ein Fehler im Hauptprogramm auftritt, dann wird eine Fehlermeldung ausgegeben und die Bearbeitung abgebrochen.

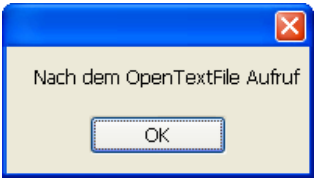
```
Set FSO = CreateObject("Scripting.FileSystemObject")

Set File = FSO.OpenTextFile("L:\Temp\Test.txt", 1, True)

MsgBox "Nach dem OpenTextFile Aufruf"
```



In dieser Variante haben Sie keine Möglichkeit, auf Fehler zu reagieren. Die einzige Möglichkeit, hier aktiv zu werden, liegt in der Einstellung „On Error Resume Next“. In diesem Fall wird mit der Bearbeitung nach einem Fehler in

der nächsten Programmzeile fortgefahren. Es wird kein Meldungsdialog für den Anwender angezeigt.

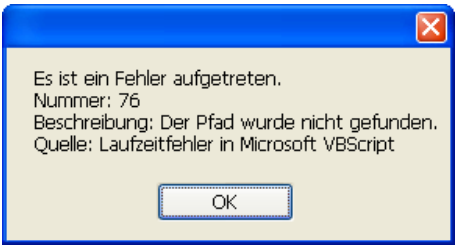
```
On Error Resume Next

Set FSO = CreateObject("Scripting.FileSystemObject")

Set File = FSO.OpenTextFile("L:\Temp\Test.txt", 1, True)

MsgBox "Nach dem OpenTextFile Aufruf"
```

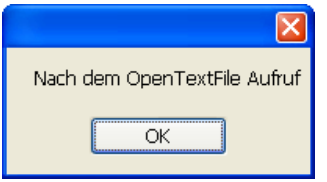


Jetzt ist der Fehler aber stillschweigend untergegangen. Wenn in diese Datei eine Quittung geschrieben werden sollte, dann fehlt diese nun und das Skript liefert nicht den geringsten Hinweis auf dieses Versäumnis.

Wenn man „Resume Next“ aktiviert, dann muss man sich auch um mögliche Fehlerzustände kümmern. Hierzu kann man die Variable Err abfragen.

```
On Error Resume Next

Set FSO = CreateObject("Scripting.FileSystemObject")

Set File = FSO.OpenTextFile("L:\Temp\Test.txt", 1, True)
If Err.Number <> 0 Then
  MsgBox "Es ist ein Fehler aufgetreten." & vbCrLf _
       & "Nummer: " & Err.Number & vbCrLf _
       & "Beschreibung: " & Err.Description & vbCrLf _
       & "Quelle: " & Err.Source
End If

MsgBox "Nach dem OpenTextFile Aufruf"
```





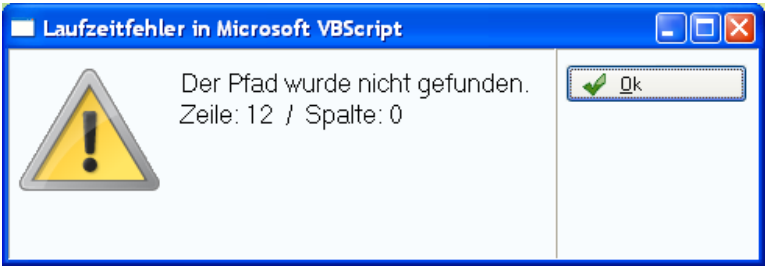
Je nach Art des Fehlers kann man in dem Skript nun geeignet reagieren. In diesem Beispiel könnte man vom Anwender einen anderen Pfad abfragen.

Wenn die Fehlerbehandlung einmal aktiviert wurde, kann man sie auch wieder abschalten. Das geschieht über den Befehl „On Error Goto 0“.

```
Set FSO = CreateObject("Scripting.FileSystemObject")

On Error Resume Next
Set File = FSO.OpenTextFile("L:\Temp\Test.txt", 1, True)
If Err.Number <> 0 Then
  MsgBox "Es ist ein Fehler im Hauptprogramm aufgetreten."
End If

MsgBox "Nach dem ersten OpenTextFile Aufruf"

On Error Goto 0
Set File = FSO.OpenTextFile("L:\Temp\Test.txt", 1, True)
If Err.Number <> 0 Then
  MsgBox "Es ist noch ein Fehler im Hauptprogramm aufgetreten."
End If

MsgBox "Nach dem zweiten OpenTextFile Aufruf"
```

Für den ersten OpenTextFile Aufruf wurde die Fehlerbehandlung in der dritten Zeile aktiviert, es kommt die entsprechende Fehlermeldung. In der

Zeile 11 wird diese aber abgeschaltet, so dass die zweite Fehlermeldung nicht kommt. Beim zweiten OpenTextFile Versuch wird das Skript abgebrochen.



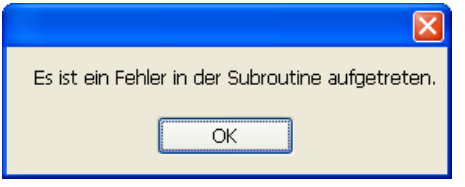
Fehlerbehandlung in Unterprogrammen

Sie können das Kommando „On Error Resume Next“ auch in Unterfunktionen setzen. In diesem Fall ist diese Einstellung nur bis zum Ende der Unterfunktion aktiv.

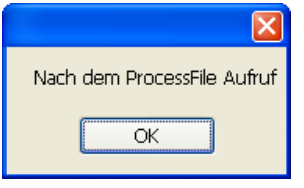
```
Set FSO = CreateObject("Scripting.FileSystemObject")

ProcessFile

MsgBox "Nach dem ProcessFile Aufruf"
```

```
Set File = FSO.OpenTextFile("L:\Temp\Test.txt", 1, True)
If Err.Number <> 0 Then
  MsgBox "Es ist ein Fehler im Hauptprogramm aufgetreten."
End If

MsgBox "Nach dem OpenTextFile Aufruf"


Sub ProcessFile
  On Error Resume Next
  Set File = FSO.OpenTextFile("L:\Temp\Test.txt", 1, True)
  If Err.Number <> 0 Then
    MsgBox "Es ist ein Fehler in der Subroutine aufgetreten."
  End If
End Sub
```

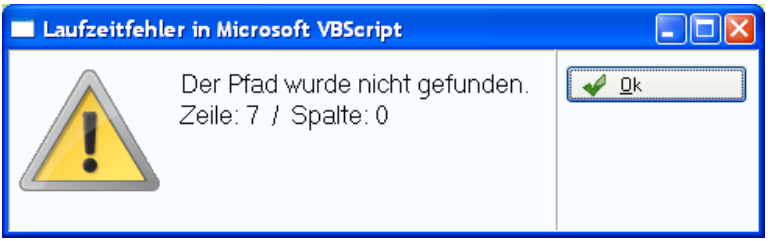
Zuerst wird die Subroutine ProcessFile aufgerufen. Dort wird die Fehlerbehandlung aktiviert. Der nun ausgelöste Dateifehler wird deshalb abgefangen und das Skript kann eine Fehlermeldung ausgeben.



Anschließend wurde die Subroutine verlassen und das Programm läuft normal weiter.



In dem weiteren Verlauf des Hauptprogramms tritt nun wieder der Dateifehler auf. Da die Fehlerbehandlung, die in der Subroutine eingeschaltet wurde, auf diese beschränkt bleibt, ist sie nun nicht mehr aktiv. Es wird also ein Programmabbruch ausgelöst.


Laufzeitfehler in Microsoft VBScript
Der Pfad wurde nicht gefunden.
Zeile: 7 / Spalte: 0
Ok

Dokumente auf Unterordner aufteilen

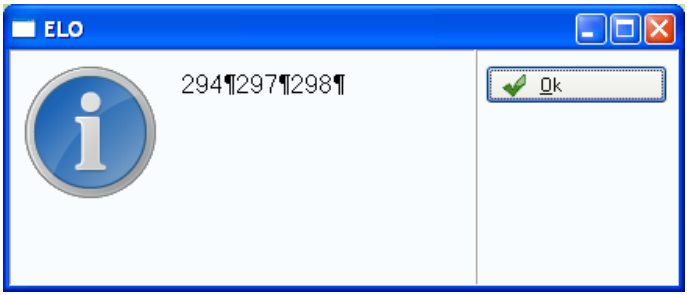
Ein Archiv wächst im Laufe der Zeit immer weiter an. Mitunter stellt man dabei fest, dass ein Ordner übermäßig viele Dokumente enthält und der Zugriff darauf unübersichtlich und langsam geworden ist. In diesem Fall kann man die Dokumente manuell auf Unterordner verteilen – was aber eine langwierige und langweilige Aufgabe ist. Falls die Unterordner durch eine alphabetische Sortierung aus der Kurzbezeichnung oder einer Indexzeile gebildet werden können, sollte diese Arbeit von einem Skript ausgeführt werden.

Das Skript muss dazu alle Dokumente aus dem aktuellen Ordner sammeln. Aus jedem Eintrag wird der erste Buchstabe der Kurzbezeichnung für den gewünschten Unterordner gelesen und der Unterordner angelegt falls er noch nicht vorhanden ist. Anschließend wird das Dokument per InsertRef Befehl in den Unterordner verschoben.

Das Anlegen eines Unterordners kann man recht einfach über den Befehl CreateStructure ausführen. Dieser Befehl erhält als Parameter einen Ordnerpfad. Er prüft nach, ob es diese Ordner bereits gibt und legt sie bei Bedarf neu an. Als Rückgabewert kommt eine Liste der ELO Elementnummern für diesen Pfad.

```
Option Explicit
Dim ELO, StartId
Set ELO = CreateObject("ELO.office")

StartId = ELO.GetEntryId(-1)
If StartId > 0 Then
  Create(StartId)
End If


Sub Create(StartId)
  Dim SubPath, Structure
  SubPath = "Ebene 1¶ Ebene 2¶ Ebene 3"
  Structure = ELO.CreateStructure(SubPath, 0 - StartId)
  Call ELO.MsgBox(Structure, "ELO", vbOkOnly)
```

End Sub



Der Rückgabewert weist die etwas unglückliche Eigenart aus, dass der letzte Ordner noch mal mit einem Trennsymbol versehen ist. Wenn man diese Liste mit **Folders = Split(FolderList, " ¶")** trennt, liegt der Zielordner also nicht in **Folders(UBound(Folders))** sondern eine Ebene tiefer.

Nachdem der Order erststellt wurde, wird das Dokument per InsertRef in diesen Ordner vorschoben.

Call ELO.InsertRef(ObjId, StartId, NewParent, 1)

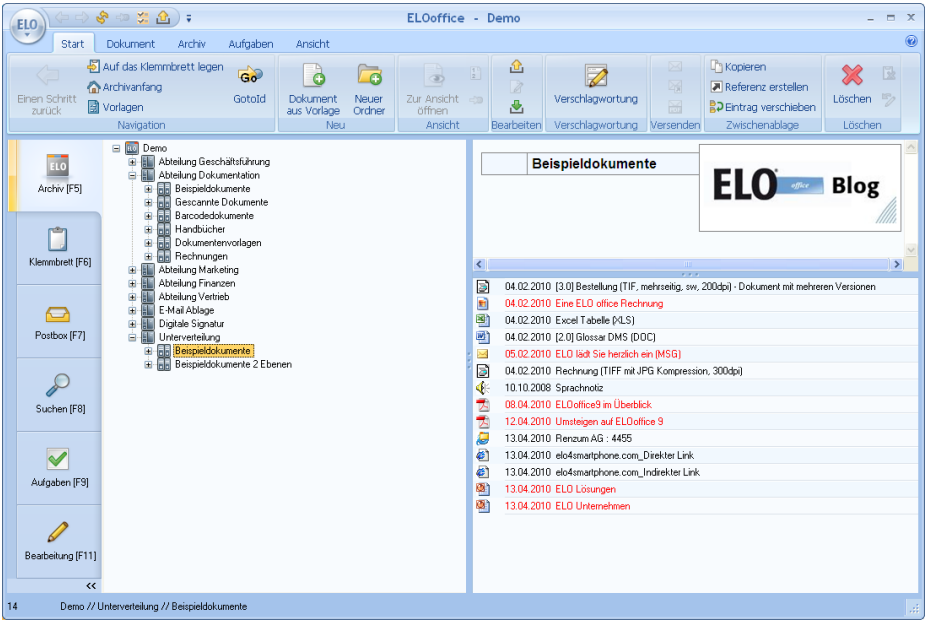
Der erste Parameter enthält die Eintragsnummer des zu verschiebenden Dokuments, der zweite Parameter den aktuellen Ablageordner, der dritte Parameter die Nummer des neuen Zielordners, der zuvor mit CreateStructure ermittelt oder erzeugt wurde. Der letzte Parameter bestimmt, ob vor dem Einfügen die Berechtigung geprüft wird.

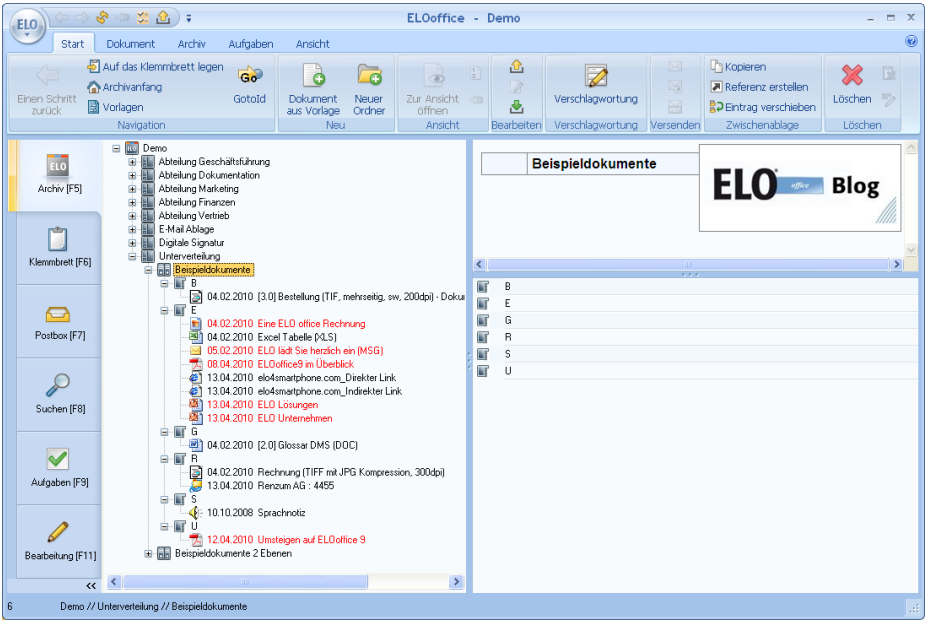
Das komplette Script sieht dann so aus:

```
' Skript: Aufteilen auf Unterordner
' Autor: Matthias Thiele
' Erstellt: 28.06.2010
' Letzte Änderung: 29.06.2010
'
' Dieses Skript läuft über alle Untereinträge des
' aktuellen Ordners und verteilt Sie auf einen
' neu erzeugte A..Z Unterordnersatz
```

```
Option Explicit


Dim ELO, StartId
Set ELO = CreateObject("ELO.office")

StartId = ELO.GetEntryId(-1)
If StartId > 0 Then
  MoveEntries(StartId)
End If


Sub MoveEntries(StartId)
  Dim EntryList, Entries, i, ObjId, Name, Msg
  Dim Folders, Structure, NewParent, LastFolder

  EntryList = ELO.CollectChildList(StartId)
  Entries = Split(EntryList, ":")

  On Error Resume Next
  For i = 0 to UBound(Entries)
    ObjId = CLng(Entries(i))
    If Err.Number <> 0 Then
      Msg = Msg & "Error: " & Entries(i) & vbCrLf
    Else
      If ELO.PrepareObjectEx(ObjId, 0, 0) > 0 Then
        If ELO.ObjTypeEx > 32 Then
          Name = ELO.ObjShort
          Structure = ELO.CreateStructure(Left(Name, 1), 0 - StartId)
          Folders = Split(Structure, "¶")
          LastFolder = Folders(UBound(Folders) - 1)
          Msg = Msg & LastFolder & " : " & Name & vbCrLf

          NewParent = CLng(LastFolder)
          If Err.Number = 0 Then
            Call ELO.InsertRef(ObjId, StartId, NewParent, 1)
          End If
        End If
      End If
    End If
  Next

  MsgBox Msg
```

End Sub

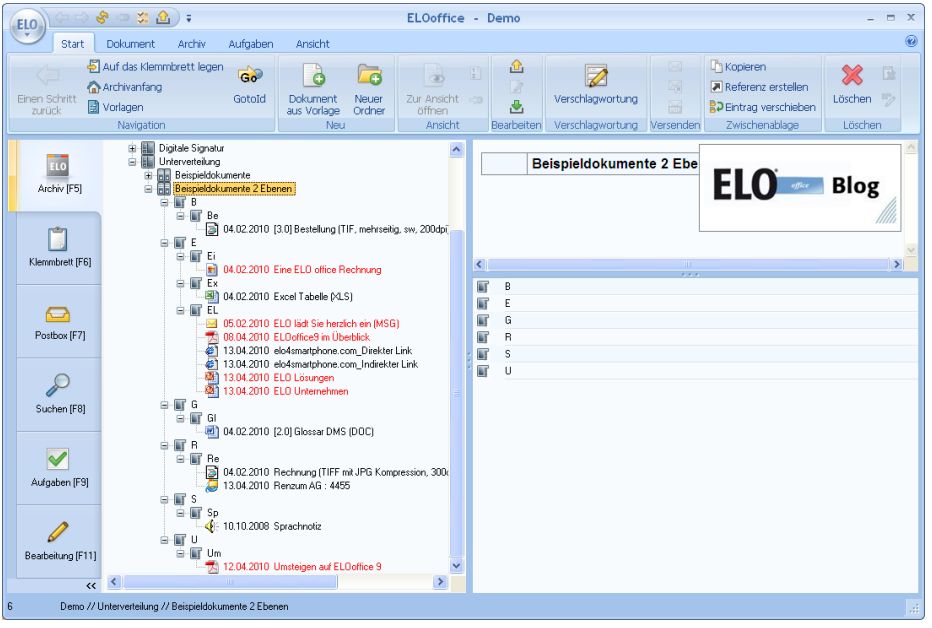

ELOoffice - Demo
Start
Dokument
Archiv
Aufgaben
Ansicht
Auf das Klemmbrett legen
Archivanfang
Vorlagen
GotoId
Navigation
Dokument aus Vorlage
Neuer Ordner
Neu
Zur Ansicht öffnen
Ansicht
Bearbeiten
Verschlagwortung
Verschlagwortung
Versenden
Kopieren
Referenz erstellen
Eintrag verschieben
Zwischenablage
Löschen
Löschen
Einen Schritt zurück
Archiv [F5]
Klemmbrett [F6]
Postbox [F7]
Suchen [F8]
Aufgaben [F9]
Bearbeitung [F11]
Demo
Abteilung Geschäftsführung
Abteilung Dokumentation
Beispieldokumente
Gescannte Dokumente
Barcodedokumente
Handbücher
Dokumentenvorlagen
Rechnungen
Abteilung Marketing
Abteilung Finanzen
Abteilung Vertrieb
E-Mail Ablage
Digitale Signatur
Unterverteilung
Beispieldokumente
Beispieldokumente 2 Ebenen
Beispieldokumente
ELO office Blog
04.02.2010 [3.0] Bestellung (TIF, mehrseitig, sw, 200dpi) - Dokument mit mehreren Versionen
04.02.2010 Eine ELO office Rechnung
04.02.2010 Excel Tabelle (XLS)
04.02.2010 [2.0] Glossar DMS (DOC)
05.02.2010 ELO lädt Sie herzlich ein (MSG)
04.02.2010 Rechnung (TIFF mit JPG Kompression, 300dpi)
10.10.2008 Sprachnotiz
08.04.2010 ELOoffice9 im Überblick
12.04.2010 Umsteigen auf ELOoffice 9
13.04.2010 Renzum AG : 4455
13.04.2010 elo4smartphone.com_Direkter Link
13.04.2010 elo4smartphone.com_Indirekter Link
13.04.2010 ELO Lösungen
13.04.2010 ELO Unternehmen
14
Demo // Unterverteilung // Beispieldokumente


ELOoffice - Demo
Start
Dokument
Archiv
Aufgaben
Ansicht
Auf das Klemmbrett legen
Archivanfang
Vorlagen
GotoId
Navigation
Dokument aus Vorlage
Neuer Ordner
Neu
Zur Ansicht öffnen
Ansicht
Bearbeiten
Verschlagwortung
Verschlagwortung
Versenden
Kopieren
Referenz erstellen
Eintrag verschieben
Zwischenablage
Löschen
Löschen
Einen Schritt zurück
Archiv [F5]
Klemmbrett [F6]
Postbox [F7]
Suchen [F8]
Aufgaben [F9]
Bearbeitung [F11]
Demo
Abteilung Geschäftsführung
Abteilung Dokumentation
Abteilung Marketing
Abteilung Finanzen
Abteilung Vertrieb
E-Mail Ablage
Digitale Signatur
Unterverteilung
Beispieldokumente
B
04.02.2010 [3.0] Bestellung (TIF, mehrseitig, sw, 200dpi) - Dokur
E
04.02.2010 Eine ELO office Rechnung
04.02.2010 Excel Tabelle (XLS)
05.02.2010 ELO lädt Sie herzlich ein (MSG)
08.04.2010 ELOoffice9 im Überblick
13.04.2010 elo4smartphone.com_Direkter Link
13.04.2010 elo4smartphone.com_Indirekter Link
13.04.2010 ELO Lösungen
13.04.2010 ELO Unternehmen
G
04.02.2010 [2.0] Glossar DMS (DOC)
R
04.02.2010 Rechnung (TIFF mit JPG Kompression, 300dpi)
13.04.2010 Renzum AG : 4455
S
10.10.2008 Sprachnotiz
U
12.04.2010 Umsteigen auf ELOoffice 9
Beispieldokumente 2 Ebenen
Beispieldokumente
ELO office Blog
B
E
G
R
S
U
6
Demo // Unterverteilung // Beispieldokumente

Falls extrem viele Dokumente in einem Ordner liegen, kann es sinnvoll sein, gleich zwei Unterebenen (AA … ZZ) einzuführen. Das Skript lässt sich leicht darauf hin anpassen.

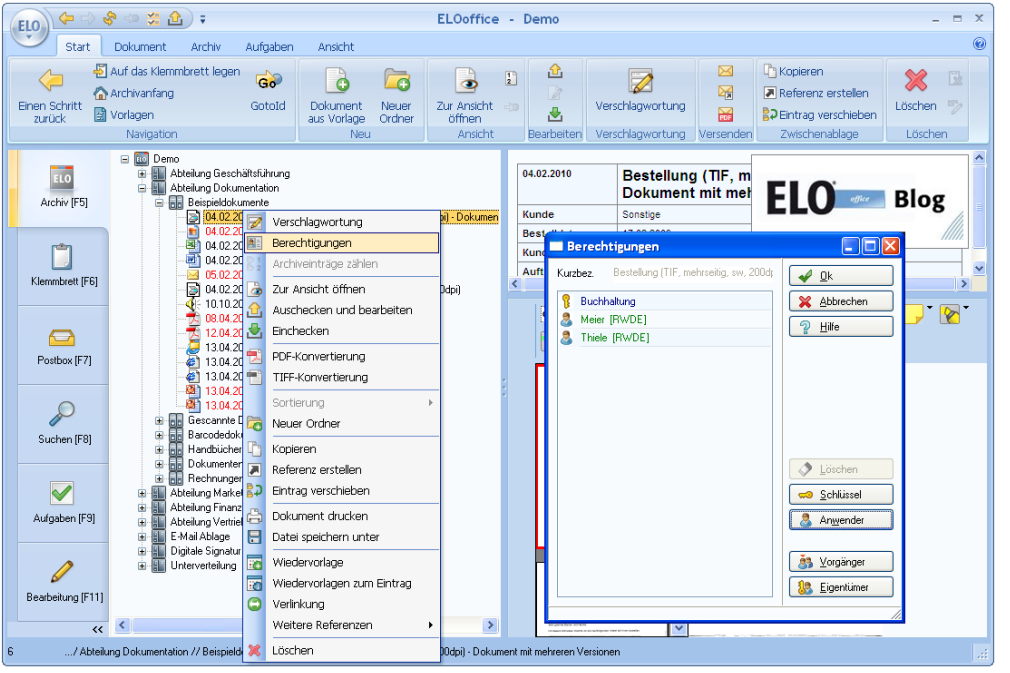
```
' Skript: Aufteilen auf Unterordner
' Autor: Matthias Thiele
' Erstellt: 28.06.2010
' Letzte Änderung: 29.06.2010
'
' Dieses Skript läuft über alle Untereinträge des
' aktuellen Ordners und verteilt Sie auf einen
' neu erzeugte AA..ZZ Unterordnersatz


Option Explicit


Dim ELO, StartId
Set ELO = CreateObject("ELO.office")

StartId = ELO.GetEntryId(-1)
If StartId > 0 Then
  MoveEntries2(StartId)
End If


Sub MoveEntries2(StartId)
  Dim EntryList, Entries, i, ObjId, Name
  Dim Msg, Structure, NewParent, SubPath
  Dim Folders, LastFolder

  EntryList = ELO.CollectChildList(StartId)
  Entries = Split(EntryList, ":")

  On Error Resume Next
  For i = 0 to UBound(Entries)
    ObjId = CLng(Entries(i))
    If Err.Number <> 0 Then
      Msg = Msg & "Error: " & Entries(i) & vbCrLf
    Else
      If ELO.PrepareObjectEx(ObjId, 0, 0) > 0 Then
        If ELO.ObjTypeEx > 32 Then
          Name = ELO.ObjShort
          SubPath = Left(Name, 1) & "¶" & Left(Name, 2)
```

```
        Structure = ELO.CreateStructure(SubPath, 0 - StartId)
        Folders = Split(Structure, "¶")
        LastFolder = Folders(UBound(Folders) - 1)
        Msg = Msg & LastFolder & " : " & Name & vbCrLf

        NewParent = CLng(LastFolder)
        If Err.Number = 0 Then
          Call ELO.InsertRef(ObjId, StartId, NewParent, 1)
        End If
      End If
    End If
  End If
Next

 MsgBox Msg
End Sub
```



Arbeiten mit Berechtigungseinstellungen

Sie können Ordner oder Dokumente im Archiv mit Zugangseinschränkungen belegen. Wenn auf einem Element einer oder mehrere Schlüssel liegen, dann hat ein Anwender nur Zugriff auf diesen Eintrag, wenn er alle Schlüssel besitzt. Zusätzlich können berechtigte Personen direkt aufgeführt werden.

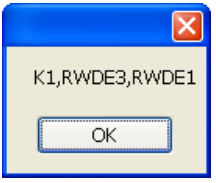


Berechtigungsliste anzeigen

Die Berechtigungen eines Eintrags können über das Property ObjAcl abgefragt werden.

```
Option Explicit
Dim ELO, StartId
Set ELO = CreateObject("ELO.office")
```

```
StartId = ELO.GetEntryId(-1)
If StartId > 0 Then
  Call ShowACL(StartId)
End If


Sub ShowACL(StartId)
  Call ELO.PrepareObjectEx(StartId, 0, 0)
  MsgBox ELO.ObjAcl
End Sub
```

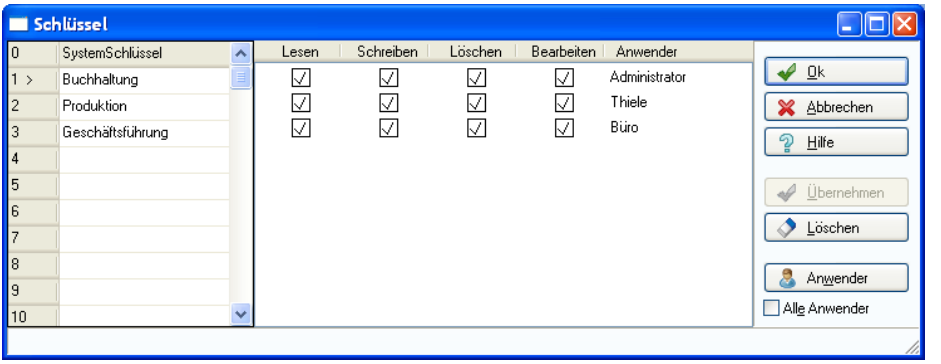


Dabei wird für jeden Eintrag in der Berechtigungsliste eine kurze Textfolge mit dem Typ und Eigentümer ausgegeben. Die unterschiedlichen Zeilen werden jeweils durch ein Komma getrennt. In diesem Beispiel enthält die ACL zuerst den Schlüssel mit der internen Schlüsselnummer 1 (K1). Danach folgt ein Vollzugriff für den Anwender mit der Nummer 3 (RWDE3) und ein Vollzugriff für den Anwender mit der Nummer 1 (RWDE1).

Die Anwenderrechte bestehen aus dem Lesezugriff (R: Read), dem Schreibzugriff (W: Write), der Löschberechtigung (D: Delete) und dem Recht zur Bearbeitung der Dokumentendatei (E: Edit).

Hinweis: Unter ELOprofessional gibt es zusätzlich noch das Recht L (List). Dieses benötigt man zum Einfügen oder Entfernen von Einträgen aus einem Ordner.

Die Schlüsselnummern kann man über den Schlüsseldialog nachschlagen. Im Skript können Sie die Schlüsselnummer aus einem Schlüsselnamen mittels LookupKeyName ermitteln. Den umgekehrten Weg geht man mittels ReadKey. Dieser Aufruf erhält eine Schlüsselnummer und liefert den dazu gehörenden Schlüsselnamen zurück.



Die Anwendernamen können mittels ReadUser und dem Property UserName ermittelt werden.

```
Option Explicit
Dim ELO, StartId
Set ELO = CreateObject("ELO.office")

StartId = ELO.GetEntryId(-1)
If StartId > 0 Then
  Call ShowACL(StartId)
End If


Sub ShowACL(StartId)
  Dim AclEntries, Entry, i, pos, Id, Msg, Access
  Call ELO.PrepareObjectEx(StartId, 0, 0)
  AclEntries = Split(ELO.ObjAcl, ",")
  For i = 0 To UBound(AclEntries)
    Entry = AclEntries(i)
    If Left(Entry, 1) = "K" Then
      Id = Mid(Entry, 2, 20)
      Msg = Msg & "Schlüssel " & Id & " : " & ELO.ReadKey(Id)
    Else
      Msg = Msg & "Anwender ( "
      For pos = 1 To Len(Entry)
        Access = Left(Entry, 1)
        Select Case Access
          Case "R": Msg = Msg & "Lesen "
          Case "W": Msg = Msg & "Schreiben "
          Case "D": Msg = Msg & "Löschen "
          Case "E": Msg = Msg & "Bearbeiten "
          Case Else
```

```
          Msg = Msg & ") " & Entry & " : "
          Call ELO.ReadUser(Entry)
          Msg = Msg & ELO.UserName
        End Select
        Entry = Mid(Entry, 2, 20)
      Next
    End If
    Msg = Msg & vbCrLf
  Next

  MsgBox Msg
End Sub
```

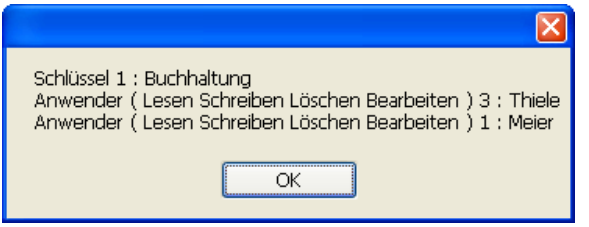
Berechtigungsliste erweitern

Das Erweitern einer bestehenden Berechtigungsliste ist noch einfacher als die Anzeige. Es muss nur der zusätzliche ACL Eintrag in Textform gebildet werden und an die ACL angehangen werden. Falls diese vorher nicht leer war, muss der neue Eintrag mit einem Komma abgetrennt werden.

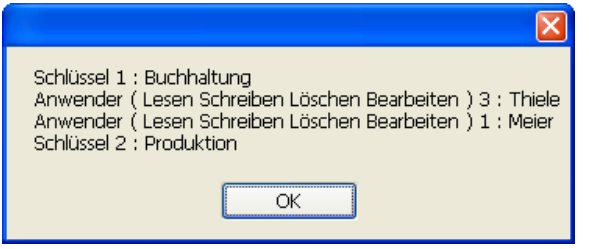
```
Option Explicit
Dim ELO, StartId
Set ELO = CreateObject("ELO.office")

StartId = ELO.GetEntryId(-1)
If StartId > 0 Then
  Call ShowACL(StartId)
  Call AddACL(StartId, 2)
  Call ShowACL(StartId)
End If
```

```
Sub AddACL(StartId, KeyNumber)
  Call ELO.PrepareObjectEx(StartId, 0, 0)
  ObjAcl = ELO.ObjAcl
  If ObjAcl <> "" Then
    ObjAcl = ObjAcl & ","
  End If

  ELO.ObjAcl = ObjAcl & "K" & KeyNumber
  Call ELO.UpdateObject
End Sub

Sub ShowACL(StartId)
  Dim AclEntries, Entry, i, pos, Id, Msg, Access
  Call ELO.PrepareObjectEx(StartId, 0, 0)
  AclEntries = Split(ELO.ObjAcl, ",")
  For i = 0 To UBound(AclEntries)
    Entry = AclEntries(i)
    If Left(Entry, 1) = "K" Then
      Id = Mid(Entry, 2, 20)
      Msg = Msg & "Schlüssel " & Id & " : " & ELO.ReadKey(Id)
    Else
      Msg = Msg & "Anwender ( "
      For pos = 1 To Len(Entry)
        Access = Left(Entry, 1)
        Select Case Access
          Case "R": Msg = Msg & "Lesen "
          Case "W": Msg = Msg & "Schreiben "
          Case "D": Msg = Msg & "Löschen "
          Case "E": Msg = Msg & "Bearbeiten "
          Case Else
            Msg = Msg & ") " & Entry & " : "
            Call ELO.ReadUser(Entry)
            Msg = Msg & ELO.UserName
        End Select
        Entry = Mid(Entry, 2, 20)
      Next
    End If
    Msg = Msg & vbCrLf
  Next

  MsgBox Msg
```

End SubVor dem AddACL:



Nach dem AddACL:



Unsichtbare Suche im Hintergrund

Wenn man in einer Ereignisbearbeitung mehrere Dokumente zusammensuchen muss, kann man die Suche verwenden. Das hat aber den Nachteil, dass man damit die Trefferliste in der Suchansicht überschreibt. Wenn die Bearbeitung von dort aus aufgerufen wurde, wird der Anwender nachhaltig verwirrt, da seine Daten unerklärlicherweise verschwunden sind. Aus diesem Grund gibt es zusätzlich noch eine unsichtbare Suche. Diese verhält sich genauso wie die normale Suche, sie wird aber nicht mit DoSearchEx sondern mit DoInvisibleSearch aufgerufen.

Als Beispiel soll bei der Rechnungsbearbeitung geprüft werden, ob zu einem eingetragenen Kunden und Rechnungsbetrag eine passende Bestellung mit gleichem Kunden und Bestellwert vorliegt. Die Rechnungsmaske im Demo Archiv hat die Nummer 8, Betrag und Kunde liegen in den Indexzeilen 1 und 2 vor (bitte daran denken: Indexzeilen werden von 0 aus gezählt). Die Bestellmaske hat die Nummer 9, der Bestellwert und der Kunde liegen in den Indexzeilen 3 und 0.

Wenn der Anwender die Betrags oder Kundenzeile verlässt und in beide Zeilen Werte eingetragen wurden, soll die Prüfroutine aufgerufen werden:

```
If ((ActionKey = 2001) Or (ActionKey = 2002)) And (Elo.ObjMaskNo = 8)
Then
  Kunde = ELO.GetObjAttrib(2)
  Betrag = ELO.GetObjAttrib(1)
  If (Kunde <> "") And (Betrag <> "") Then
    ItemsFound = SearchValues(Kunde, Betrag)
```

In der Prüfroutine wird zuerst die aktuelle Verschlagwortung gesichert, sonst würden Sie mit der Suche die Daten der Rechnung überschreiben. Anschließend wird mit **PrepareObjectEx(0, 0, 9)** eine leere Bestellung erzeugt und Betrag und Kunde durch **SetObjAttrib()** in die Indexzeilen geschrieben. Die Funktion DoInvisibleSearch liefert die Anzahl der Treffer wieder, diese wird als Rückgabewert der Funktion verwendet.

```
Function SearchValues(Kunde, Betrag)
  SearchValues = 0
  Call ELO.SaveObject(1)
  Call ELO.PrepareObjectEx(0, 0, 9)
  Call ELO.SetObjAttrib(0, Kunde)
  Call ELO.SetObjAttrib(3, Betrag)
  SearchValues = ELO.DoInvisibleSearch
  Call ELO.SaveObject(2)
End Function
```

Wenn die Suche keine Treffer gefunden hat, gibt das Skript eine Meldung aus, dass keine passende Bestellung gefunden werden konnte.

```
   ItemsFound = SearchValues(Kunde, Betrag)
   If ItemsFound < 1 Then
     Call ELO.MsgBox("Zum Kunden " & Kunde & " liegt keine" _
               & vbCrLf & "Bestellung mit einem Wert von " _
               & Betrag & " EUR vor.", "ELO", vbOkOnly)
   End If
```

Das komplette Skript, welches als Ereignisaufruf „Beim Bearbeiten der Verschlagwortung“ eingetragen werden muss, sieht dann so aus:

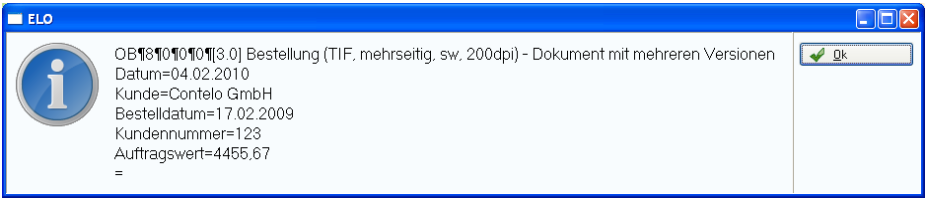
```
' Skript: EVT_Bestellungsprüfung
' Autor: Matthias Thiele
' Erstellt: 29.06.2010
' Letzte Änderung: 29.06.2010
'
' Dieses Skript wird als Ereignisroutine
' "Beim Bearbeiten der Verschlagwortung"
```

```
' angemeldet. Wenn ein Rechungsdokument
' (Masken-Id = 8) einen Kunden- und Wert-
' Eintrag hat, wird in den Bestellungen
' nach dieser Kombination gesucht. Falls
' diese nicht gefunden wird, meldet das
' Skript, dass eine unbekannte Bestellung
' vorliegt


option explicit

Dim ELO, ActionKey, Kunde, Betrag, ItemsFound
Set ELO = CreateObject("ELO.office")

ActionKey = ELO.ActionKey
If ((ActionKey = 2001) Or (ActionKey = 2002)) And (Elo.ObjMaskNo = 8)
Then
  Kunde = ELO.GetObjAttrib(2)
  Betrag = ELO.GetObjAttrib(1)
  If (Kunde <> "") And (Betrag <> "") Then
    ItemsFound = SearchValues(Kunde, Betrag)
    If ItemsFound < 1 Then
      Call ELO.MsgBox("Zum Kunden " & Kunde & " liegt keine" _
                & vbCrLf & "Bestellung mit einem Wert von " _
                & Betrag & " EUR vor.", "ELO", vbOkOnly)
    End If
  End If
End If


Function SearchValues(Kunde, Betrag)
  SearchValues = 0
  Call ELO.SaveObject(1)
  Call ELO.PrepareObjectEx(0, 0, 9)
  Call ELO.SetObjAttrib(0, Kunde)
  Call ELO.SetObjAttrib(3, Betrag)
  SearchValues = ELO.DoInvisibleSearch
  Call ELO.SaveObject(2)
End Function
```

Wenn mehr als eine Bestellung gefunden wurde, kann man eine Liste der Bestelldatumseinträge ausgeben. Diese Ausgabe erfolgt nach dem SearchValues Aufruf, wenn die Anzahl in ItemsFound größer als 1 ist.

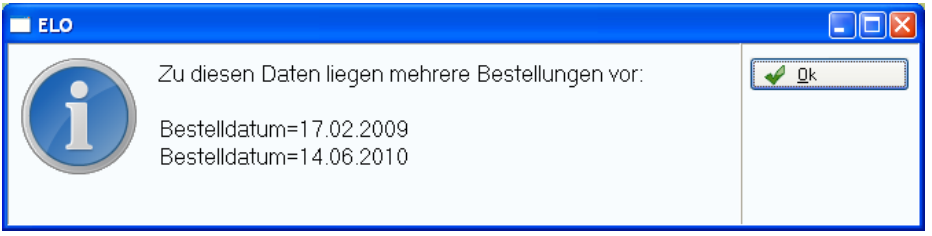
```
If ItemsFound > 1 Then
  Call ShowAllItems(ItemsFound)
End If
```

In der Funktion ShowAllItems wird die unsichtbare Trefferliste mittels GetListEntry ausgelesen. Diese enthält neben ein paar Basisdaten (Eintragsnummer, Typ, Kurzbezeichnung nun mehr) auch alle Indexzeilen.



Im Beispiel ist nur die vierte Zeile (Index 3) mit dem Bestelldatum interessant. Diese wird durch den Split Befehl erreichbar.

```
Sub ShowAllItems(ItemsFound)
  Dim i, Msg, Entry, Items

  For i = 0 To ItemsFound - 1
    Entry = ELO.GetListEntry(i)
    Items = Split(Entry, vbLf)
    If (UBound(Items) > 3) Then
      Msg = Msg & Items(3) & vbCrLf
    End If
  Next

  Call ELO.MsgBox("Zu den Daten liegen mehrere Bestellungen vor: "_
            & vbCrLf & vbCrLf & Msg, "ELO", vbOkOnly)
End Sub
```



Arrays Sortieren

In vielen Skripten müssen Daten sortiert werden die in einem Array liegen. Dazu gibt es eine Vielzahl von Beispielen. In den meisten ist das eigentliche Sortierkriterium aber fest vorgegeben. Selbst wenn nur einfache Strings sortiert werden sollen, gibt es in der Praxis einige Möglichkeiten: soll die Groß/ Kleinschreibweise berücksichtigt werden oder nicht? Enthält der Text ein Datum im TT.MM.JJJJ Format (dann ist eine reine Textsortierung nicht sinnvoll, weil der 12.09.2010 vor dem 22.08.2010 kommt)? Beginnt der Text mit einer Zahl (dann kommt die 12 vor der 9 bei einer einfachen Textsortierung)? Man müsste also für jede spezielle Suchstrategie eine Kopie der Sortierroutine erstellen und dort die jeweilig benötigten Vergleichsfunktionen einfügen. Das führt zu unnötig viel und unübersichtlichen Skriptcode. Aber es geht auch besser:

Statt für jede spezielle Situation eine Kopie der kompletten Sortierroutine zu erstellen, wird nur eine spezielle Vergleichsfunktion geschrieben. Diese wird dann einer universellen Sortierroutine als Parameter mitgegeben.

Zuerst werden also die Vergleichsfunktionen erstellt. Im Folgenden eine nicht „case sensitive“ Vergleichsfunktion CompareIsLowerU – dort wird vor dem Vergleich jeweils ein UpperCase durchgeführt, so dass nicht zwischen Klein/ Großschreibweise unterschieden wird. Und eine „case sensitive“ Vergleichsfunktion CompareIsLowerCase – dort wird direkt verglichen. Mit der Folge, dass es eine A..Za..z Sortierung gibt.

```
Function CompareIsLowerU(Val1, Val2)
  CompareIsLowerU = StrComp(Val1, Val2, vbTextCompare) < 0
End Function

Function CompareIsLowerCase(Val1, Val2)
  CompareIsLowerIC = Val1 < Val2
End Function
```

Der Aufruf der Sortierfunktion erhält dann als Parameter das zu sortierende Array, die Nummer des ersten und letzten Eintrags und einen Funktionszeiger auf die gewünschte Vergleichsfunktion.

```
Call Sort(Data, 0, UBound(Data), GetRef("CompareIsLowerCase"))
```

Hinweis: Prinzipiell hätte man die Nummern für den ersten und letzten Eintrag nicht als Parameter übergeben müssen, da die Sortierfunktion diese Werte auch selber ermitteln könnte. Praktisch sind diese aber dann, wenn nur Teile eines Arrays sortiert werden sollen.

Sortierung einfacher Datentypen

Die eigentliche Sortierfunktion ist ein einfacher QuickSort. Dieser Algorithmus ist in der Literatur oft und ausführlich beschrieben, so dass darauf hier nicht weiter eingegangen werden soll.

```
Function Sort(ByRef Data, ByRef Low, ByRef High, ByRef
CompareIsLower)
  Dim MidVal, Swap, CurLow, CurHigh, Midpoint

  If High <= Low Then
    Exit Function
  End If

  CurLow = Low
  CurHigh = High
  Midpoint = (Low + High) \ 2
  MidVal = Data(Midpoint)

  Do While (CurLow <= CurHigh)
    Do While CompareIsLower(Data(CurLow), MidVal)
      CurLow = CurLow + 1
      If CurLow = High Then
        Exit Do
      End If
    Loop

    Do While CompareIsLower(MidVal, Data(CurHigh))
```

```
    CurHigh = CurHigh - 1
    If CurHigh = Low Then
      Exit Do
    End If
  Loop

  If (CurLow <= CurHigh) Then
    Swap = Data(CurLow)
    Data(CurLow) = Data(CurHigh)
    Data(CurHigh) = Swap

    CurLow = CurLow + 1
    CurHigh = CurHigh - 1
  End If

 Loop

 If Low < CurHigh Then
   Call Sort(Data, Low, CurHigh, CompareIsLower)
 End If

 If CurLow < High Then
   Call Sort(Data, CurLow,High, CompareIsLower)
 End If
End Function
```

Ein Beispiel zur Verwendung der Sortierfunktion liest zuerst die Kurzbezeichnung aller Einträge des aktuell ausgewählten Ordners ein und sortiert diese dann in alphabetischer Reihenfolge.

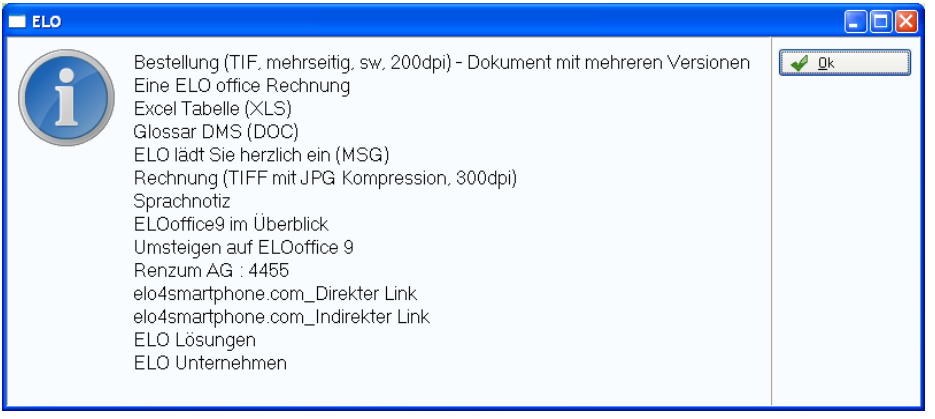
```
' Skript: Quicksort
' Autor: Matthias Thiele
' Erstellt: 29.06.2010
' Letzte Änderung: 30.06.2010
'
' Dieses Programm zeigt die Verwendung
' der Quicksort Bibliotheksfunktion.


Option Explicit
Dim ELO, FSO, StartId, Data, ChildList, Id
Set ELO = CreateObject("ELO.office")
```

```
StartId = ELO.GetEntryId(-1)
If StartId > 0 Then
  ' Zuerst wird der Ordner eingelesen und angezeigt
  Call LoadData(StartId)
  Call ShowData

  ' Anschließend werden die Daten sortiert
  Call Sort(Data, 0, UBound(Data), GetRef("CompareIsLowerNoCase"))

  ' Zum Abschluss wird die sortierte Liste angezeigt
  Call ShowData
End If


' Die Subroutine LoadData liest alle
' Kurzbezeichnungen innerhalb des aktuellen
' Ordners in das Array Data ein.
Sub LoadData(StartId)
  Dim ChildList, Items, i

  ChildList = ELO.CollectChildList(StartId)
  Items = Split(ChildList, ":")
  Redim Data (UBound(Items) - 1)

  For i = 0 To UBound(Items) - 1
    Data(i) = ELO.GetEntryName(Items(i))
  Next
End Sub


' Die Subroutine ShowData zeigt die Liste der
' Einträge aus dem Array Data in einer
' MessageBox an.
Sub ShowData
  Dim i, Msg

  For i = 0 to UBound(Data)
    Msg = Msg & Data(i) & vbCrLf
  Next

  Call ELO.MsgBox(Msg, "ELO", vbOkOnly)
End Sub

' Hier folgen die Vergleichsfunktionen, es
' wird das Namensschema CompareIsLower*
```

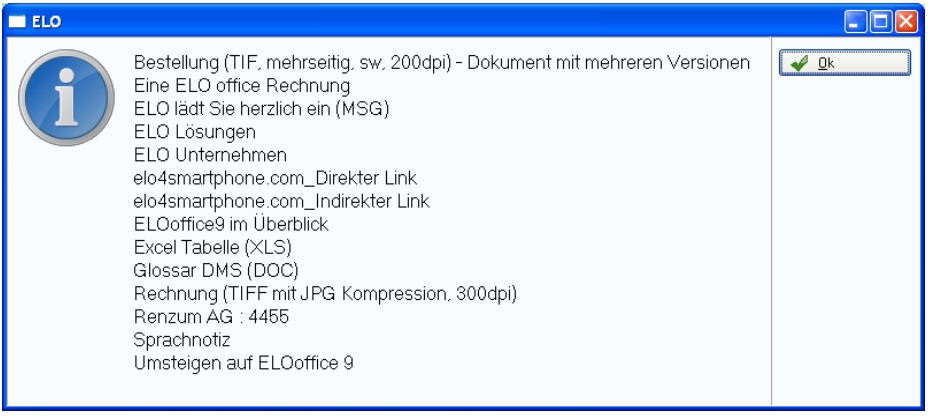
```
' verwendet.

' Die Vergleichsfunktion "NoCase" führt
' einen case insensitiven Vergleich aus.
' Das ergibt für Texte eine natürliche
' Sortierung
Function CompareIsLowerNoCase(Val1, Val2)
  CompareIsLowerNoCase = StrComp(Val1, Val2, vbTextCompare) < 0
End Function

' Die Vergleichsfunktion "Case" führt
' einen binären Vergleich aus. Das hat zur
' folge, dass Kleinbuchstaben komplett hinter
' den Großbuchstaben liegen A..Za..z
Function CompareIsLowerCase(Val1, Val2)
  CompareIsLowerCase = Val1 < Val2
End Function


' Die Sortierfunktion Sort erhält als Parameter
' das zu sortierende Array, erstes und letztes
' zu sortierende Element und die Vergleichsfunktion
Function Sort(ByRef Data, ByRef Low, ByRef High, ByRef
CompareIsLower)
  Dim MidVal, Swap, CurLow, CurHigh, Midpoint

  If High <= Low Then
    Exit Function
  End If

  CurLow = Low
  CurHigh = High
  Midpoint = (Low + High) \ 2
  MidVal = Data(Midpoint)

  Do While (CurLow <= CurHigh)
    Do While CompareIsLower(Data(CurLow), MidVal)
      CurLow = CurLow + 1
      If CurLow = High Then
        Exit Do
      End If
    Loop

    Do While CompareIsLower(MidVal, Data(CurHigh))
      CurHigh = CurHigh - 1
```

```
    If CurHigh = Low Then
      Exit Do
    End If
  Loop

  If (CurLow <= CurHigh) Then
    Swap = Data(CurLow)
    Data(CurLow) = Data(CurHigh)
    Data(CurHigh) = Swap

    CurLow = CurLow + 1
    CurHigh = CurHigh - 1
  End If

 Loop

 If Low < CurHigh Then
  Call Sort(Data, Low, CurHigh, CompareIsLower)
 End If

 If CurLow < High Then
  Call Sort(Data, CurLow,High, CompareIsLower)
 End If
End Function
```

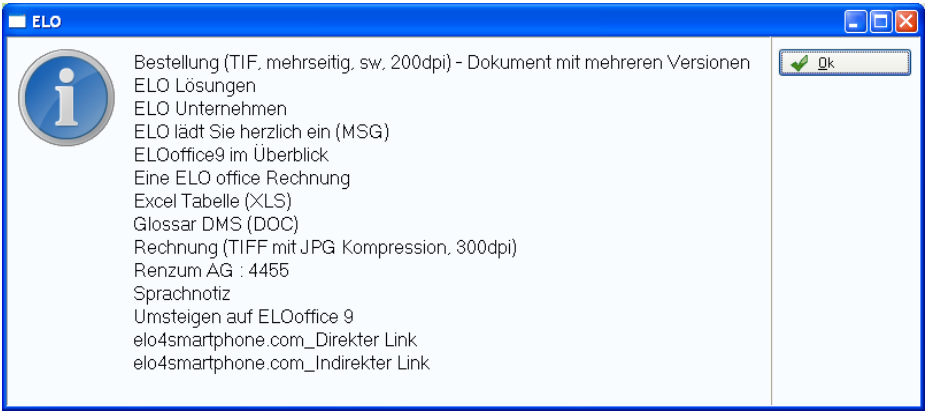
Anzeige der eingelesenen Liste



Anzeige der Liste nach einer Textsortierung (nicht case sensitive). Die kleingeschriebenen Einträge elo4smartphone sind normal in die Liste einsortiert.



Anzeige der Liste nach einer binären Sortierung. Die kleingeschriebenen Einträge elo4smartphone sind ganz am Ende der Liste.



Sortierung komplexer Strukturen

Das Beispiel sortiert eine einfache Textliste, in der Praxis sind die zu sortierenden Strukturen aber oft komplexer und bestehen aus mehreren Teilen. Das folgende Beispiel soll für die Verarbeitung nicht nur den Namen

sondern auch das Dokumentendatum und die ELO Eintragsnummer benötigen. Um das Beispiel noch etwas zu erweitern, soll die Sortierung nun primär nach dem Dokumentendatum erfolgen und nur bei gleichem Datum eine Untersortierung nach dem Namen durchgeführt werden.

Der erste Gedanke wäre nun vielleicht die Verwendung vor drei Arrays, eines für den Namen, das zweite für das Dokumentendatum und das dritte für die Eintragsnummer. Dann wird es aber bei der Sortierung schwierig. Wenn nun das Array Namen sortiert wird, dann passt die Reihenfolge nicht mehr zu den beiden anderen Arrays.

Hier müsste also die Sortierfunktion so angepasst werden, dass mit dem austauschen der Inhalte im Namensarray die gleichen Zeilen in den beiden anderen Arrays identisch vertauscht werden. Das ist umständlich und Fehlerträchtig. Zudem wird anhand des Programmcodes der logische Zusammenhang einer Zeile in den drei unterschiedlichen Arrays nicht sichtbar.

Zum Glück liefert die Programmierung mit Klassen hier eine bessere Lösung. Für die Daten wird eine Klasse angelegt, die den Namen, das Dokumentendatum und die Eintragsnummer zusammen speichert. Für diese Properties werden Zugriffsfunktionen zum setzen und lesen der Werte definiert.

```
Class ItemStore
  Private M_Name
  Private M_ObjId
  Private M_DocDate

  Public Property Get Name
    Name = M_Name
  End Property

  Public Property Let Name(NewName)
    M_Name = NewName
  End Property

  Public Property Get ObjId
```

```
    ObjId = M_ObjId
  End Property

  Public Property Let ObjId(NewObjId)
    M_ObjId = NewObjId
  End Property

  Public Property Get DocDate
    DocDate = M_DocDate
  End Property

  Public Property Let DocDate(NewDocDate)
    M_DocDate = NewDocDate
  End Property

End Class
```

Beim Einlesen der Ordnerliste wird nun für jeden Eintrag so ein Objekt erzeugt, die Properties gefüllt und dieses Objekt in die Data Liste eingefügt.

```
Sub LoadData(StartId)
  Dim ChildList, Items, i

  ChildList = ELO.CollectChildList(StartId)
  Items = Split(ChildList, ":")
  Redim Data (UBound(Items) - 1)

  For i = 0 To UBound(Items) - 1
    If ELO.PrepareObjectEx(Items(i), 0, 0) > 0 Then
      Set Data(i) = New ItemStore
      Data(i).Name = ELO.ObjShort
      Data(i).DocDate = ELO.ObjXDate
      Data(i).ObjId = ELO.GetEntryId(-2)
    End If
  Next
End Sub
```

Die zweistufige Sortierung kann man nun durch eine entsprechende Vergleichsfunktion durchführen.

```
Function CompareIsLowerNoCase(Val1, Val2)
  If Val1.DocDate < Val2.DocDate Then
```

```
    CompareIsLowerNoCase = False
  Else
    If Val1.DocDate = Val2.DocDate Then
      CompareIsLowerNoCase = StrComp(Val1.Name, _
                  Val2.Name, vbTextCompare) < 0
    Else
      CompareIsLowerNoCase = True
    End If
  End If
End Function
```

Leider wird das so einfach nicht funktionieren. Das Dokumentendatum hat keine richtige „natürliche“ Sortierung, da es im Textformat vorliegt. In diesem Fall ist der 12.09.2010 vor dem 21.01.2010, das ist aber falsch. In der Vergleichsfunktion müsste man die Jahres-, Monats- und Tagesanteile des Datums einzeln betrachten. Das ist Aufwändig und benötigt viel Zeit, da die Vergleichsfunktion extrem oft aufgerufen wird. In solchen Fällen kann es sinnvoll sein, schon beim Füllen des Objekts eine geeignete Darstellung für den Vergleich zu erzeugen. Dafür wird ein weiteres internes Feld „SortInfo“ in der Klasse angelegt. Dieses wird aus den anderen Feldern automatisch gefüllt. Dazu wird das Datum zuerst in ein ISO Datum JJJJMMTT umgewandelt und anschließend der Name angefügt. Hierdurch ergibt sich eine natürliche textuelle Sortierung, so dass die Vergleichsfunktion sehr einfach gehalten werden kann.

```
  Private Sub FillSortInfo
    If Len(M_DocDate) = 10 Then
      M_SortInfo = Right(M_DocDate, 4) & _
                Mid(M_DocDate, 4, 2) & _
                Left(M_DocDate, 2) & _
                M_Name
    End If
  End Sub
```

Die Vergleichsfunktion reduziert sich auf einen einfachen binären Vergleich, da die SortInfo schon passend aufbereitet wurde.

```
Function CompareIsLower(Val1, Val2)
```

```
  CompareIsLower = Val1.SortInfo < Val2.SortInfo
End Function
```

Bei der ItemStore Klasse ist nun noch zu Beachten, dass beim Setzen des Namens oder Datums das SortInfo Property aktualisiert wird. Die komplette Klasse sieht dann so aus:

```
Class ItemStore
  Private M_Name
  Private M_SortInfo
  Private M_ObjId
  Private M_DocDate

  Public Property Get Name
    Name = M_Name
  End Property

  Public Property Let Name(NewName)
    M_Name = NewName
    Call FillSortInfo
  End Property

  Public Property Get ObjId
    ObjId = M_ObjId
  End Property

  Public Property Let ObjId(NewObjId)
    M_ObjId = NewObjId
  End Property

  Public Property Get DocDate
    DocDate = M_DocDate
  End Property

  Public Property Let DocDate(NewDocDate)
    M_DocDate = NewDocDate
    Call FillSortInfo
  End Property

  Public Property Get SortInfo
    SortInfo = M_SortInfo
  End Property
```

```
  Private Sub FillSortInfo
    If Len(M_DocDate) = 10 Then
      M_SortInfo = Right(M_DocDate, 4) & _
                   Mid(M_DocDate, 4, 2) & _
                   Left(M_DocDate, 2) & _
                   LCase(M_Name)
    End If
  End Sub
End Class
```

Leider kann die Sort Funktion nicht sofort für das Objekt übernommen werden. Bei der Zuweisung von Objekten ist ein führendes Set notwendig, bei einfachen Datentypen ist es nicht erlaubt.

```
Set Swap = Data(CurLow)
Set Data(CurLow) = Data(CurHigh)
Set Data(CurHigh) = Swap
```

> Hinweis: Die Version im Anhang zur Verwendung als Library in eigenen Skripten hat eine zusätzliche Kontrolle für den Datentyp und kann deshalb gleichermaßen für Objekte wie für einfache Datentypen verwendet werden.

Das komplette Beispiel sieht dann so aus:

```
' Skript: Quicksort2
' Autor: Matthias Thiele
' Erstellt: 30.06.2010
' Letzte Änderung: 30.06.2010
'
' Dieses Programm zeigt die Verwendung
' der Quicksort Bibliotheksfunktion.


Option Explicit
Dim ELO, FSO, StartId, Data, ChildList, Id
Set ELO = CreateObject("ELO.office")

StartId = ELO.GetEntryId(-1)
If StartId > 0 Then
```

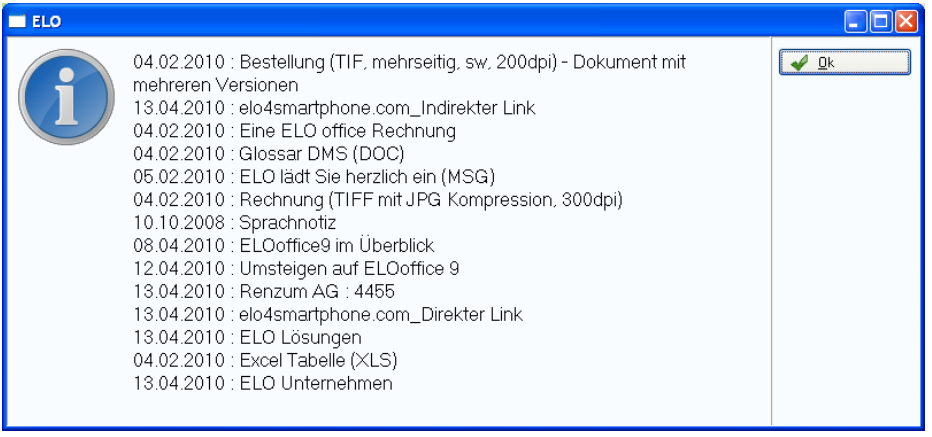
```
  ' Zuerst wird der Ordner eingelesen und angezeigt
  Call LoadData(StartId)
  Call ShowData

  ' Anschließend werden die Daten sortiert
  Call Sort(Data, 0, UBound(Data), GetRef("CompareIsLower"))

  ' Zum Abschluss wird die sortierte Liste angezeigt
  Call ShowData
End If


' Die Subroutine LoadData liest alle
' Kurzbezeichnungen innerhalb des aktuellen
' Ordners in das Array Data ein.
Sub LoadData(StartId)
  Dim ChildList, Items, i

  ChildList = ELO.CollectChildList(StartId)
  Items = Split(ChildList, ":")
  Redim Data (UBound(Items) - 1)

  For i = 0 To UBound(Items) - 1
    If ELO.PrepareObjectEx(Items(i), 0, 0) > 0 Then
      Set Data(i) = New ItemStore
      Data(i).Name = ELO.ObjShort
      Data(i).DocDate = ELO.ObjXDate
      Data(i).ObjId = ELO.GetEntryId(-2)
    End If
  Next
End Sub


' Die Subroutine ShowData zeigt die Liste der
' Einträge aus dem Array Data in einer
' MessageBox an.
Sub ShowData
  Dim i, Msg

  For i = 0 to UBound(Data)
    Msg = Msg & Data(i).DocDate & " : " & Data(i).Name & vbCrLf
  Next

  Call ELO.MsgBox(Msg, "ELO", vbOkOnly)
End Sub
```

```
' Die Vergleichsfunktion führt einen
' einfachen binären Vergleich aus. Das
' ist möglich, da die SortInfo bereits
' passend aufbereitet wurde.
Function CompareIsLower(Val1, Val2)
  CompareIsLower = Val1.SortInfo < Val2.SortInfo
End Function


' Die Sortierfunktion Sort erhält als Parameter
' das zu sortierende Array, erstes und letztes
' zu sortierende Element und die Vergleichsfunktion
Function Sort(ByRef Data, ByRef Low, ByRef High, ByRef
CompareIsLower)
  Dim MidVal, Swap, CurLow, CurHigh, Midpoint

  If High <= Low Then
    Exit Function
  End If

  CurLow = Low
  CurHigh = High
  Midpoint = (Low + High) \ 2
  Set MidVal = Data(Midpoint)

  Do While (CurLow <= CurHigh)
    Do While CompareIsLower(Data(CurLow), MidVal)
      CurLow = CurLow + 1
      If CurLow = High Then
        Exit Do
      End If
    Loop

    Do While CompareIsLower(MidVal, Data(CurHigh))
      CurHigh = CurHigh - 1
      If CurHigh = Low Then
        Exit Do
      End If
    Loop

    If (CurLow <= CurHigh) Then
      Set Swap = Data(CurLow)
      Set Data(CurLow) = Data(CurHigh)
      Set Data(CurHigh) = Swap
```

```
      CurLow = CurLow + 1
      CurHigh = CurHigh - 1
    End If

  Loop

  If Low < CurHigh Then
    Call Sort(Data, Low, CurHigh, CompareIsLower)
  End If

  If CurLow < High Then
    Call Sort(Data, CurLow,High, CompareIsLower)
  End If
End Function

Class ItemStore
  Private M_Name
  Private M_SortInfo
  Private M_ObjId
  Private M_DocDate

  Public Property Get Name
    Name = M_Name
  End Property

  Public Property Let Name(NewName)
    M_Name = NewName
    Call FillSortInfo
  End Property

  Public Property Get ObjId
    ObjId = M_ObjId
  End Property

  Public Property Let ObjId(NewObjId)
    M_ObjId = NewObjId
  End Property

  Public Property Get DocDate
    DocDate = M_DocDate
  End Property

  Public Property Let DocDate(NewDocDate)
    M_DocDate = NewDocDate
```

```
    Call FillSortInfo
  End Property

  Public Property Get SortInfo
    SortInfo = M_SortInfo
  End Property

  Private Sub FillSortInfo
    If Len(M_DocDate) = 10 Then
      M_SortInfo = Right(M_DocDate, 4) & _
                   Mid(M_DocDate, 4, 2) & _
                   Left(M_DocDate, 2) & _
                   LCase(M_Name)
    End If
  End Sub
End Class
```

In dieser Version werden in der MessageBox sowohl das Dokumentendatum wie auch die Kurzbezeichnung ausgegeben.



Vor der Sortierung sind sowohl die Datumseinträge wie auch die Namen in zufälliger Reihenfolge.

Nach der Sortierung sind die Einträge primär nach dem Dokumentendatum sortiert. Alle Dokumente mit gleichem Datum sind nach dem Namen untersortiert.

Properties in eigenen Klassen

Die Klasse ItemStore verwendet für den Zugriff auf die Variablen für Namen oder Dokumentendatum eigens erstellte Zugriffsfunktionen.

```
Class ItemStore
  Private M_Name

  Public Property Get Name
    Name = M_Name
  End Property

  Public Property Let Name(NewName)
    M_Name = NewName
  End Property
```

Dieser Aufwand mag auf den ersten Blick überflüssig erscheinen, man könnte doch einfach mit einer öffentlichen Variablen arbeiten:

```
Class ItemStore
  Public Name
```

Der Nutzen wird es etwas später klar – wenn das Feld für die SortInfo eingeführt wird. Wenn sich der Name oder das Dokumentendatum ändert, dann muss auch die SortInfo angepasst werden. Beim direkten Zugriff auf die Variablen müsste der Skriptentwickler selber darauf achten, dass diese aktualisiert wird. Wenn aber Zugriffsfunktionen verwendet werden, kann das die Klasse intern automatisch durchführen.

```
Public Property Let Name(NewName)
  M_Name = NewName
  Call FillSortInfo
End Property
```

Sobald im Skript ein neuer Name gesetzt wird, aktualisiert diese Funktion nicht nur den lokalen Speicher für den Namen. Es wird zusätzlich auch die SortInfo neu berechnet und ist somit immer auf den aktuellen Stand.

Properties für Objekte

Bei der Zuweisung haben wir schon gesehen, dass VB-Script zwischen einfachen Datentypen und Objekten unterscheidet. Bei Objekten ist ein Set vor der Zuweisung notwendig, bei einfachen Datentypen verboten. Diese Asymmetrie setzt sich auch bei den Properties fort. Einen Unterschied gibt es beim Setzen des Property-Wertes. Falls es sich dabei um einen einfachen Datentyp handelt, sieht es so aus:

```
Public Property Let Name(NewName)
  M_Name = NewName
End Property
```

Bei Objekten wird statt des Let ein Set benötigt:

```
Public Property set Document(NewDocument)
  Set M_Document = NewDocument
End Property
```

Aber auch beim Lesen des Property-Wertes gibt es einen kleinen Unterschied. Bei der Zuweisung des Rückgabewerts wird, wie bei Objekten allgemein üblich, ein Set vor dem Statement benötigt:

```
Public Property Get Document
  Set Document = M_Document
End Property
```

Wenn man das Set vergisst oder eines zu viel setzt, dann gibt es Laufzeitfehler. Aus diesem Grund ist es wichtig, dass man sich immer im Klaren darüber ist, ob man mit einfachen Datentypen oder Objekten arbeitet.

Initialisierung von Klassenobjekten

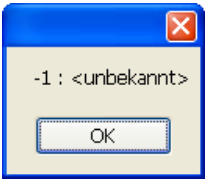
Ein praktischer Vorteil eines Objekts liegt darin, dass beim Anlegen einer neuen Instanz automatisch eine Initialisierung ausgeführt werden kann.

Für diese Aufgabe stellt VB-Script einen speziellen Methodennamen „Class_Initialize“ zur Verfügung. Wenn Sie diese Methode implementieren, können Sie hier die Befehle für die Initialisierung hinterlegen. Diese Methode wird automatisch immer ausgeführt, wenn ein neues Objekt erzeugt wird.

Im folgenden Beispiel hat jedes Objekt zwei Variablen, den Namen und die Id. Objekte, die noch keinem Anwender zugeordnet sind, sollen als Id immer eine -1 besitzen und als Name <unbekannt> enthalten. Über die Class_Initialize Methode kann sicher gestellt werden, dass das immer der Fall ist.

```
Dim Item

Set Item = New Test
MsgBox Item.AsText

Class Test
  Private Name
```

```
  Private Id

  Private Sub Class_Initialize
    Name = "<unbekannt>"
    Id = -1
  End Sub

  Public Function AsText
    AsText = Id & " : " & Name
  End Function

End Class
```



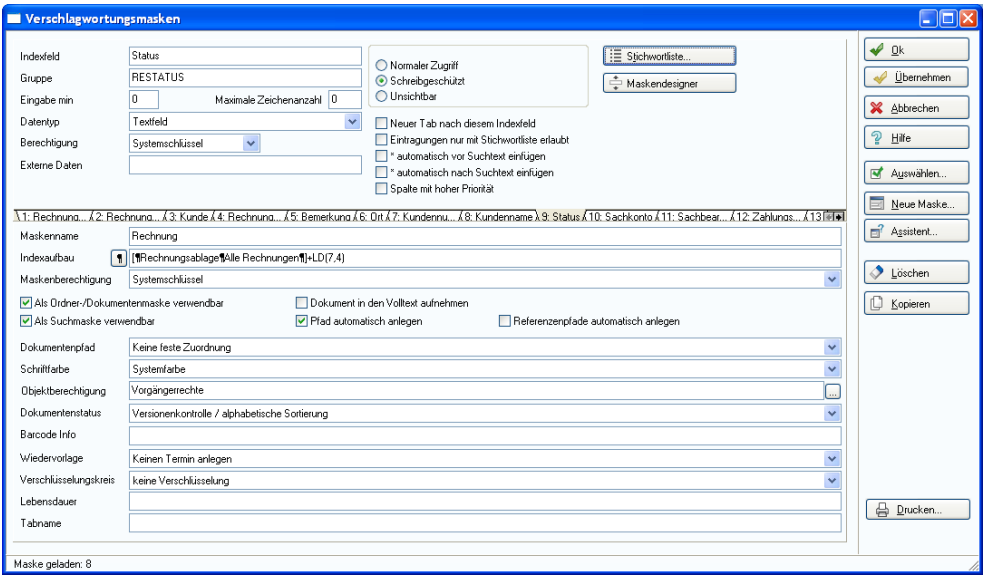
Einfacher Rechnungsworkflow

Das folgende Beispiel soll einen einfachen Rechnungsworkflow zeigen. Neue Rechnungen werden von der Buchhaltung eingescannt und im Archiv abgelegt. Zusätzlich werden die Rechnungsdaten erfasst und eingetragen, welcher Sachbearbeiter für die Zuordnung des Sachkontos und der Rechnungsfreigabe verantwortlich ist.

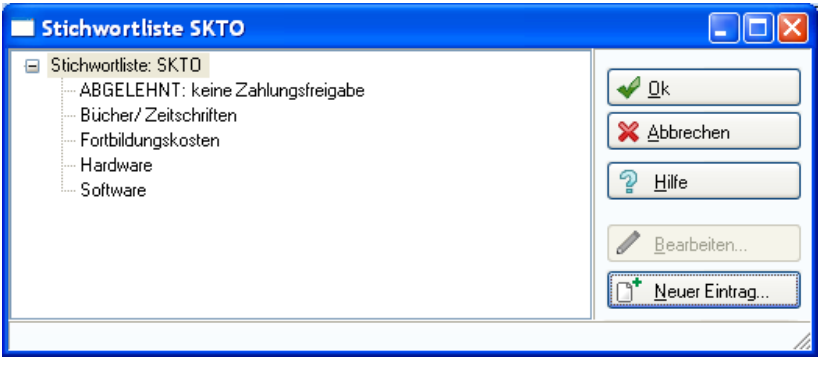
Der verantwortliche Sachbearbeiter findet die Rechnung dann in seinem persönlichen Ordner „Offene Rechnungen“ vor. Er wählt ein Sachkonto aus (oder das Konto „keine Freigabe“) und speichert den Eintrag wieder. In diesem Fall wird die Rechnung an die Buchhaltung zur Zahlung oder Klärung zurück gegeben. Die Buchhaltung findet die Rechnung dann im Ordner „Freigegeben“ oder „Abgelehnt“ vor.

Vorbereitung der Verschlagwortungsmaske und Skripte

In einem ersten Schritt muss die Rechnungsmaske um zwei Felder ergänzt werden: um eine Zeile für das Sachkonto (Gruppenname SKTO) und eine weitere Zeile für den Rechnungsstatus (RESTATUS). Das Feld für den Rechnungsstatus sollte auf „Schreibgeschützt“ gesetzt werden, es darf nur durch die Skriptprogrammierung geändert werden, nicht aber durch manuelle Eingriffe des Anwenders.

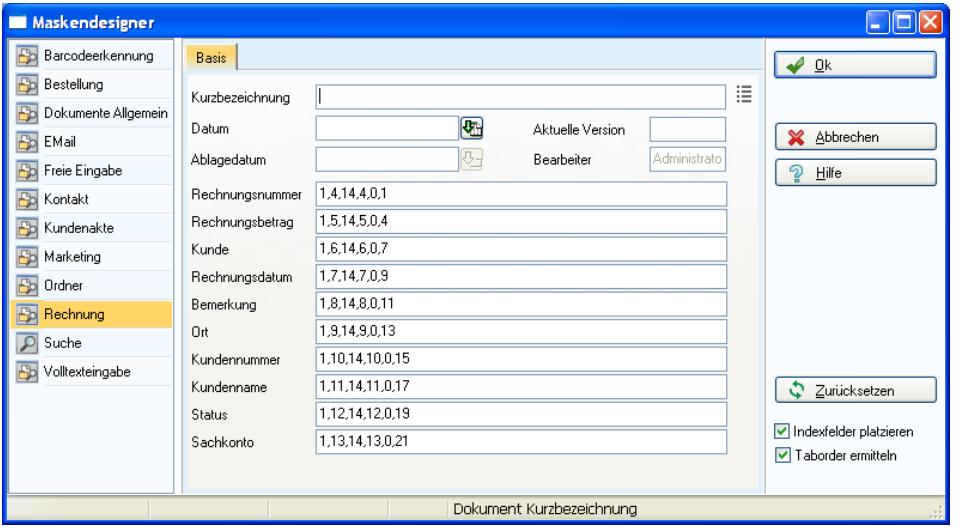


Zum Feld Sachkonto wird eine Stichwortliste mit den Sachkonten und dem „Pseudo-Konto“ Abgelehnt angelegt.

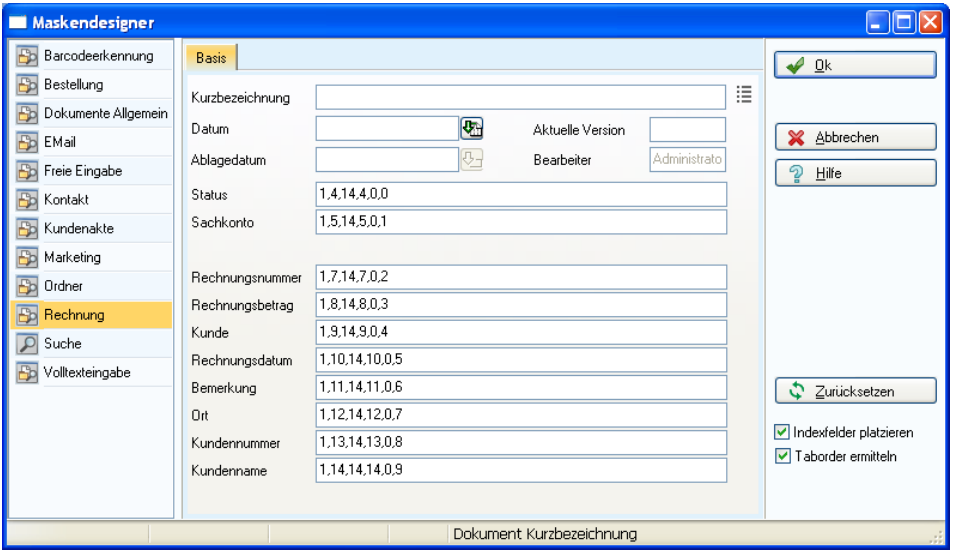


In dieser Form hat die Verschlagwortung den Nachteil, dass das Statusfeld am Ende der Liste angeordnet ist. Der Übersichtlichkeit halber wäre eine Platzierung ganz am Anfang sinnvoller. Eine Umsortierung kann im Maskendesigner erfolgen. Er kann über die Schaltfläche „Maskendesigner“ oben rechts im Dialog „Verschlagwortungsmasken“ erreicht werden. Dort kann man die Indexzeilen per Drag&Drop umsortieren. Im Beispiel werden die beiden Zeilen „Status“ und „Sachkonto“ vom Ende der Liste an den

Anfang gezogen (die anderen Zeilen müssen natürlich entsprechend nach unten verschoben werden). Eine Leerzeile erhöht zudem die Übersichtlichkeit.

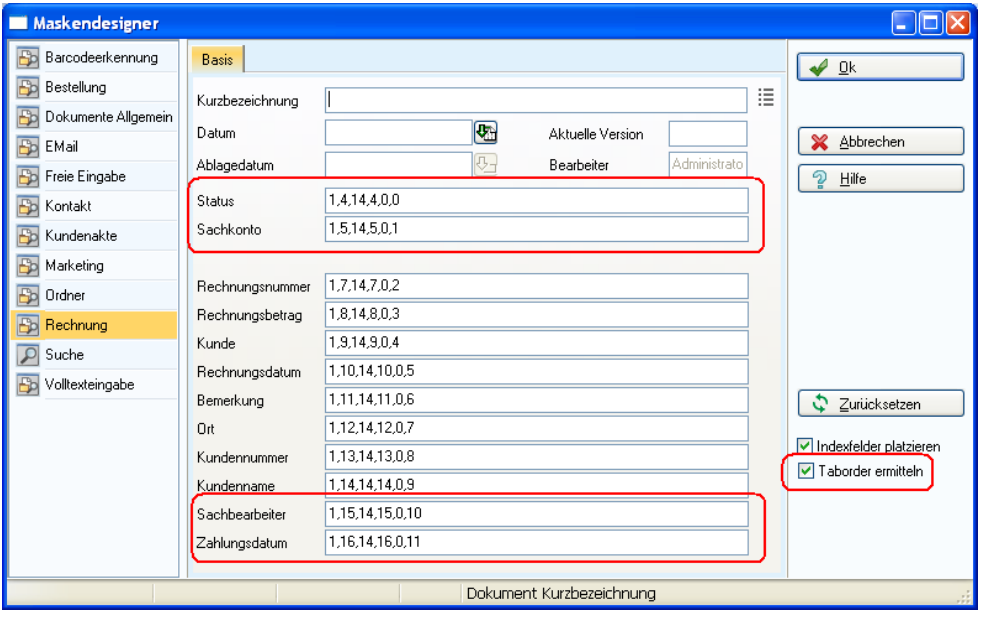


Vorher - Nachher



Weiterhin werden noch die Indexzeilen für den verantwortlichen Sachbearbeiter für die Kostenstelle und das Zahldatum der Buchhaltung benötigt.

Die Sachbearbeiter werden in einer Stichwortliste aufgeführt. Man könnte die Indexzeile für Sachbearbeiter auch als Anwenderfeld definieren, dann kann man aber beliebige Anwender als Sachbearbeiter auswählen. Wenn man eine Stichwortliste verwendet und das Kennzeichen setzt „Nur aus Stichwortliste füllen“, dann ist der Personenkreis individuell konfigurierbar. Damit das Sachbearbeiterfeld nicht leer bleibt, wird eine Mindestlänge von einem Zeichen definiert.



Hier nun die endgültige Version der Maske. Status und Sachkonto wurden nach oben verschoben, eine Leerzeile trennt sie vom Rest der Eingaben. Am Ende wurden die Eingabefelder Sachbearbeiter und Zahlungsdatum hinzugefügt. Über die Option „Taborder ermitteln“ sorgt ELO automatisch dafür, dass sich beim Durchlaufen durch die Verschlagwortungsmaske über die Tab-Taste eine natürliche Reihenfolge ergibt.

Hinweis: Das Feld für das Zahldatum ist als ISO Datum und nicht als einfaches Datumsfeld konfiguriert. Für den Anwender ergibt sich hieraus kein Unterschied, intern verursacht es eine wichtige Änderung: ein normales Datum hat keine natürliche Textsortierung - der 12.10.2010 liegt vor dem 20.01.2010. Dieses Problem führt dazu, dass normale Datumsfelder bei Bereichssuchen unerwartete Ergebnisse liefern (und aus der Sicht des Anwenders auch unsinnige Ergebnisse). Deshalb sollte man nur ISO Datumsfelder verwenden. Das normale Datumsfeld existiert nur aus Kompatibilitätsgründen zu älteren Archiven.

Für das Ablageziel wird ein Indexaufbau hinterlegt. Er sorgt dafür, dass die Buchhaltung die Rechnungen nicht manuell in einen Rechnungsordner legen muss. Die komplette Rechnungsbearbeitung findet in einem Schrank „Rechnungsablage“ statt. Dort gibt es einen Unterordner „Alle Rechnungen“, der wiederum Jahresordner mit den eigentlichen Rechnungsdokumenten enthält. Damit diese Ordner nicht von Hand erzeugt werden müssen, wird in der Maskendefinition die Option „Pfad automatisch anlegen“ aktiviert. Das Skript legt zudem später in dem Schrank Rechnungsablage automatisch weitere Ordner für die Sachbearbeiter und die Buchhaltung an.

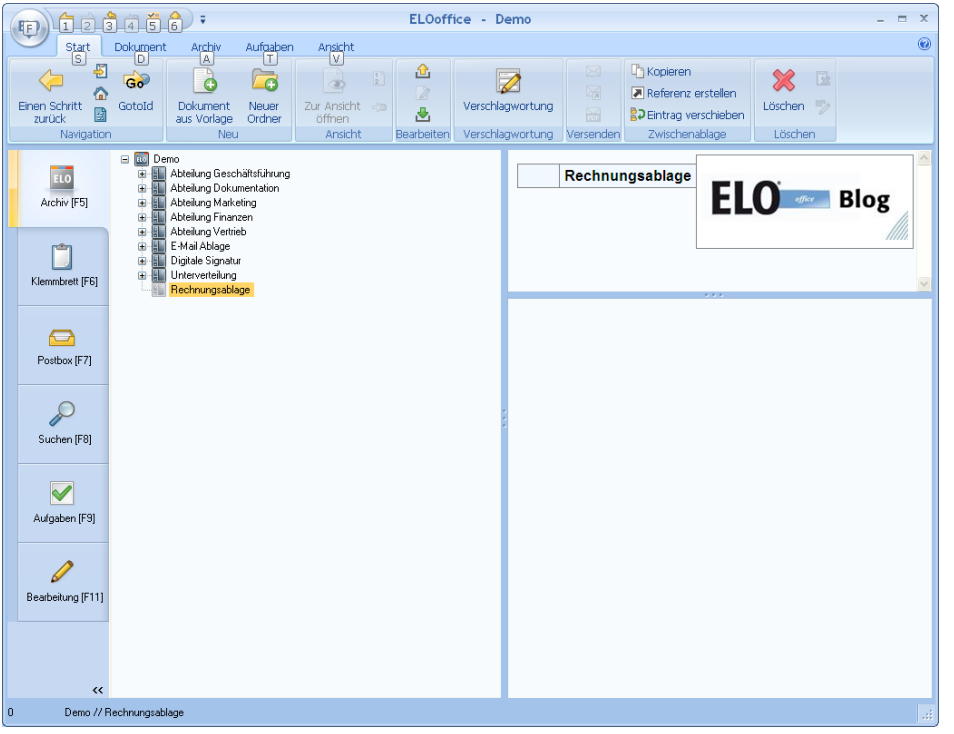
Die komplette Maskendefinition sieht dann so aus:

Rechnung [8]

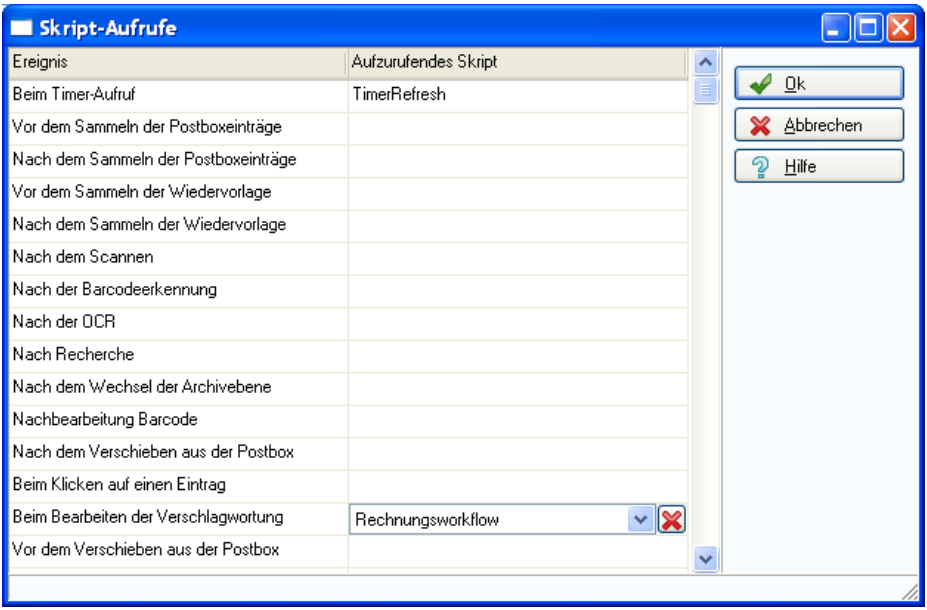
Maskenname:	Rechnung [8]
Indexaufbau:	[¶Rechnungsablage¶Alle Rechnungen¶]+LD(7,4)
Maskenberechtigung:	Systemschlüssel
Als Ordner-/Dokumentenmaske verwendbar:	ja
Als Suchmaske verwendbar:	ja
Dokument in den Volltext aufnehmen:	nein
Dokumentenpfad:	Keine feste Zuordnung
Schriftfarbe:	Systemfarbe
Objektberechtigung:	Vorgängerrechte
Dokumentenstatus:	Versionenkontrolle / alphabetische Sortierung
Barcode Info:	
Wiedervorlage:	Keinen Termin anlegen
Verschlüsselungskreis:	keine Verschlüsselung
Lebensdauer:	
Tabname:	

Nr	Index Feld	Gruppe	Min	Max	Typ	Zugriffsart	Verschliessen	Lasche	Stichwort	* vor	* nach
1	Rechnungsnumme	RE-Nummer	1	0	Txt	Normal	Systemschlüs				
2	Rechnungsbetra	RE-Betrag	1	0	Txt	Normal	Systemschlüs				
3	Kunde	Kunde	0	0	Txt	Normal	Systemschlüs		x		
4	Rechnungsdatum	RE-Datum	0	0	Dat	Normal	Systemschlüs				
5	Bemerkung	Bemerkung	0	0	Txt	Normal	Systemschlüs				
6	Ort	ORT	0	0	Txt	Normal	Systemschlüs				
7	Kundennummer	KDNR	0	0	Txt	Normal	Systemschlüs				
8	Kundenname	KDNAM	0	0	Txt	Normal	Systemschlüs				
9	Status	RESTATUS	0	0	Txt	Schreibgeschützt	Systemschlüs				
10	Sachkonto	SKTO	0	0	Txt	Normal	Systemschlüs				
11	Sachbearbeiter	REOWNER	1	0	Txt	Normal	Systemschlüs		x		
12	Zahlungsdatum	REZDAT	0	0	ISO	Normal	Systemschlüs				

Zur ersten Ablage einer Rechnung muss neben der Maskendefinition nur das Skript konfiguriert werden und der Schrank „Rechnungsablage“ angelegt werden. Alle weiteren benötigten Ordner werden durch das Skript automatisch erzeugt.

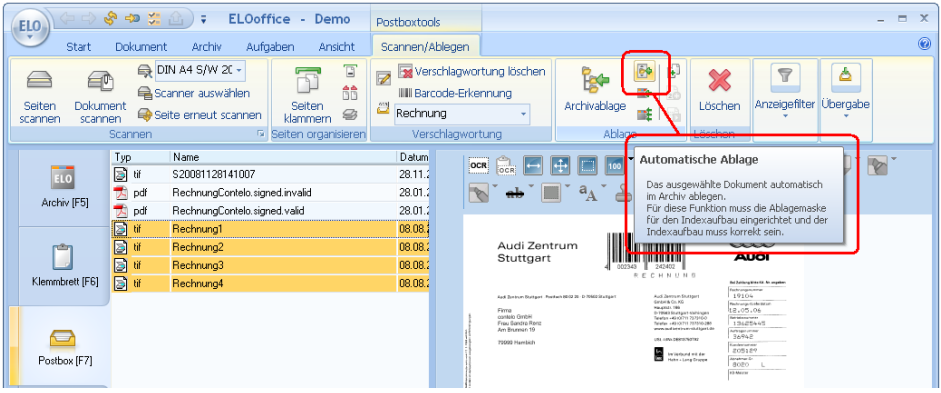


In den folgenden Zeilen wird das Skript für den Rechnungsworkflow beschrieben. Es muss im Skript Event „Beim Bearbeiten der Verschlagwortung“ hinterlegt werden. Zusätzlich gibt es ein weiteres Skript „TimerRefresh“ im Event „Beim Timer-Aufruf“. Dieses Skript sorgt für eine automatische Aktualisierung der dynamischen Register nach einer Änderung.

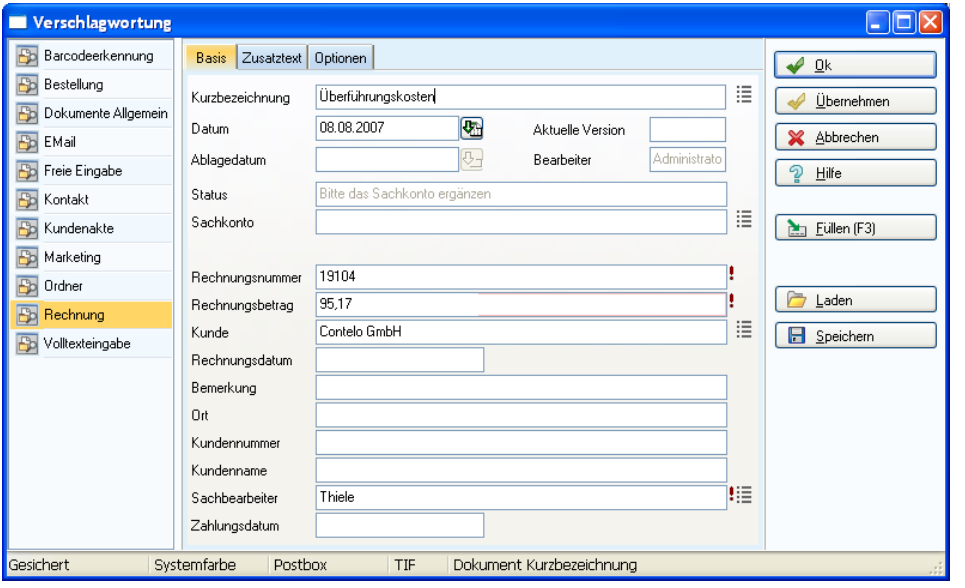


Ablegen einer Rechnung

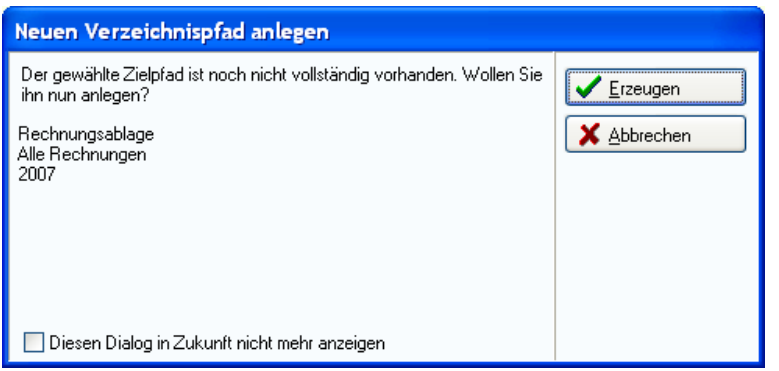
Zur Rechnungsablage müssen diese von der Buchhaltung eingescannt und bei Bedarf geklammert werden. In der Postbox können diese dann selektiert werden und über die Schaltfläche „Automatische Ablage“ in das Archiv übertragen werden.



ELOoffice arbeitet dann die markierten Einträge nacheinander ab. Über den Verschlagwortungsdialog gibt die Buchhaltung dann die Rechnungsdaten und den Sachbearbeiter an, das Programm legt die Rechnung im Zielordner ab und das Skript erstellt die notwendigen Unterordner für die Sachbearbeiter.

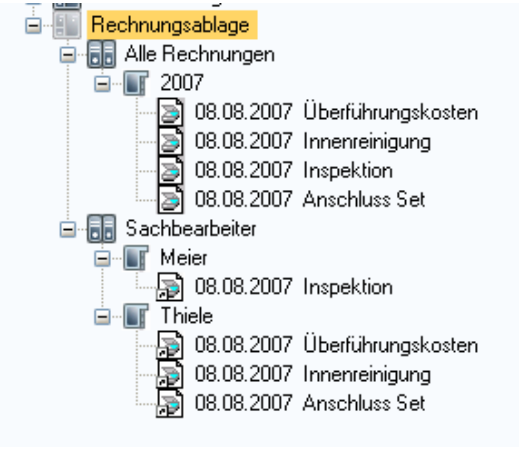


Bei der ersten Ablage fragt das System möglicherweise ab, ob der Zielordner erstellt werden soll.



Nach Abschluss der Arbeiten sind die Dokumente im Archiv verfügbar. Der Ordner „Alle Rechnungen“ enthält einen Unterordner 2007 welcher

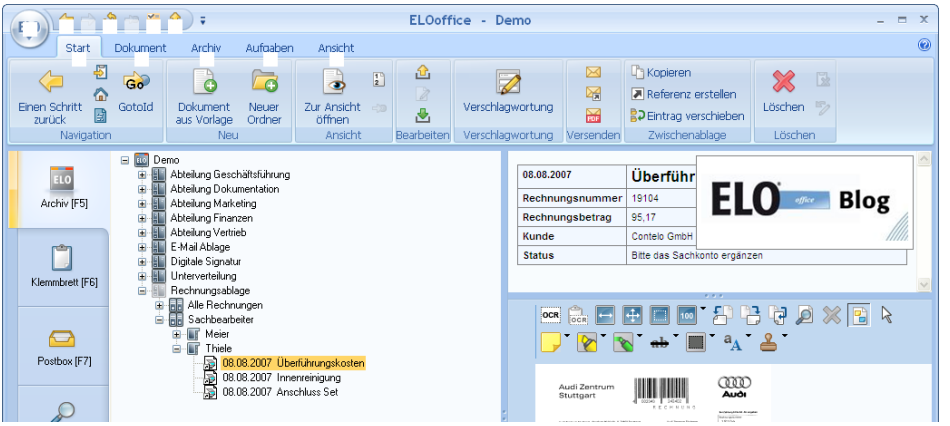
wiederum die vier abgelegten Rechnungen enthält. Die Rechnung „Inspektion“ wurde dem Sachbearbeiter „Meier“ zugeordnet. Dafür hat das Skript automatisch das dynamische Register „Sachbearbeiter \ Meier“ angelegt. Die anderen Rechnungen wurden dem Bearbeiter Thiele zugeordnet. Auch für Ihn wurde ein dynamisches Register erzeugt.



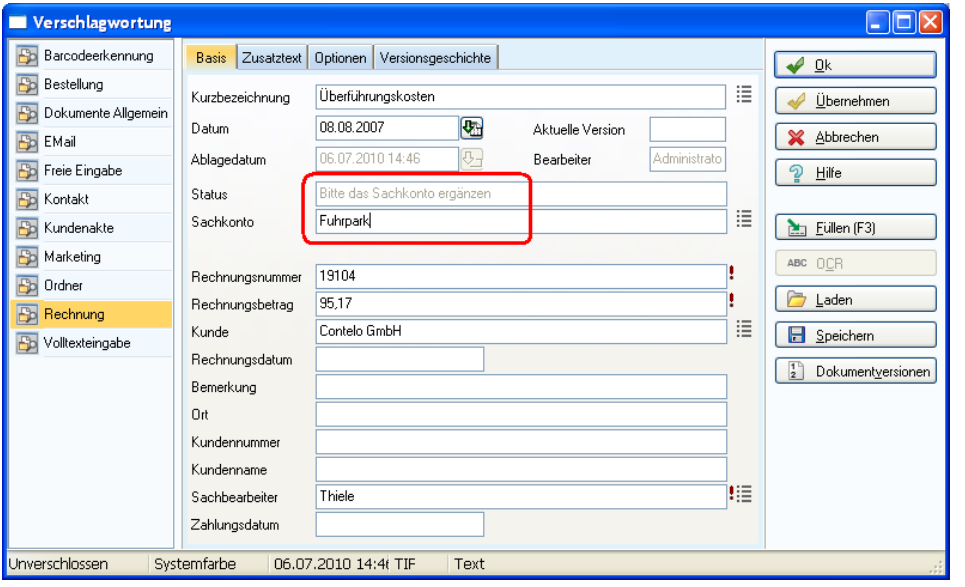
Eintragen des Sachkontos

Das Sachkonto wird im Rahmen dieses Workflows vom verantwortlichen Sachbearbeiter eingetragen. Dafür hat jeder Sachbearbeiter einen persönlichen Ordner in dem alle offenen Rechnungen aufgelistet werden. Man könnte für diese Sachbearbeiter das weiter oben aufgeführte Skript verwenden, welches beim Öffnen des Archivs gleich in einen konfigurierten Ordner verzweigt. So stellt man sicher, dass die Sachbearbeiter ihre offenen Rechnungen regelmäßig zur Kenntnis nehmen.

Für den Sachbearbeiter „Thiele“ würde sich der Startbildschirm so darstellen:



Er würde die Rechnungen jetzt der Reihe nach abarbeiten. Dafür muss er nur den Verschlagwortungsdialog aufrufen, das Sachkonto ergänzen und mittels „Ok“ abspeichern.

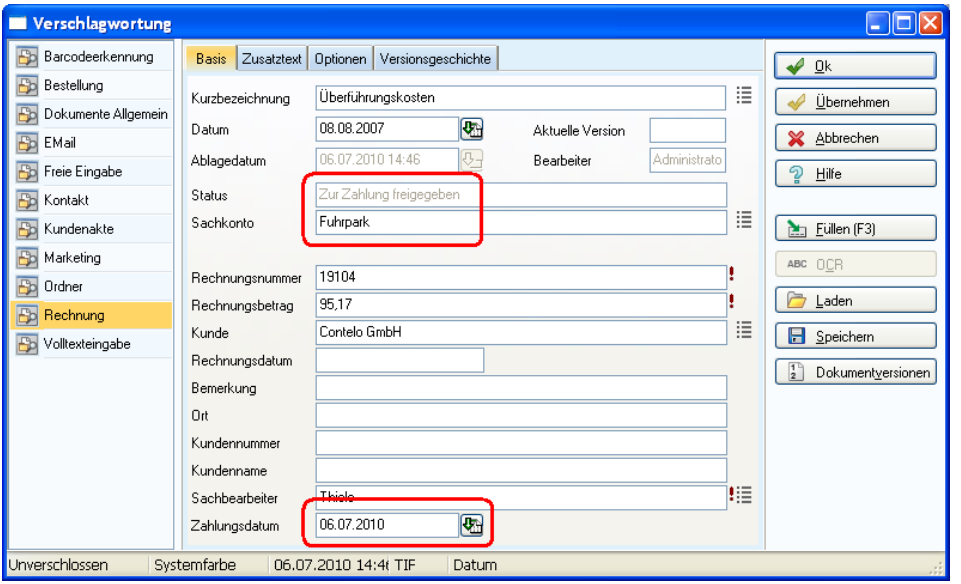


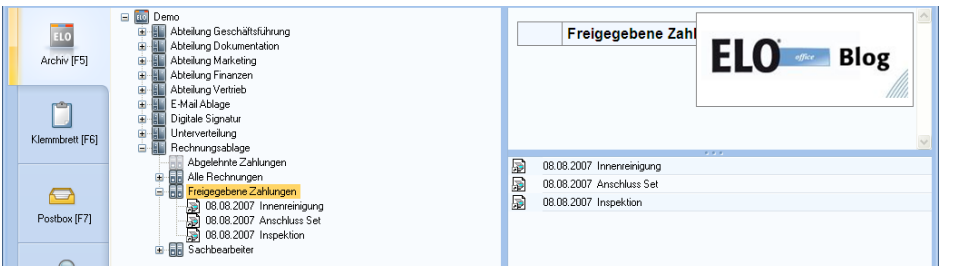
Verbuchung und Zahlung

Alle Rechnungen, die von den Sachbearbeitern frei gegeben wurden, findet die Buchhaltung im Ordner „Freigegebene Zahlungen“ vor.

Die Rechnung wird nun bezahlt und in der Verschlagwortung das Zahlungsdatum eingetragen. Beim Speichern wird der Status nun auf „Gezahlt“ umgestellt, so dass diese Rechnung nicht mehr in diesem dynamischen Register aufgeführt wird.

Hinweis: Dynamische Register werden beim Aufblättern automatisch mit den aktuellen Daten gefüllt. Sie werden bei einer Veränderung im Archiv jedoch nicht automatisch aktualisiert. Deshalb hinterlegt das Workflow Skript in einem Cookie den aktuellen Pfad. Dieser wird von dem Timer Skript aufgegriffen und der Pfad dann neu angezeigt. Im Allgemeinen sollte man jedoch mit Skripten im Timer Event extrem vorsichtig sein. Wenn sich die automatische Skriptverarbeitung mit einer Anwendereingabe überschneidet, kann es zu unerwarteten Ergebnissen führen!





Beschreibung des Workflowskripts

Das Workflowskript greift an zwei verschiedenen Zeitpunkten in die Verarbeitung ein. Zum Einen bei der Erzeugung einer neuen Rechnung. In diesem Fall wird über die Unterroutine „PrepareNew“ das Statusfeld mit „Bitte das Sachkonto ergänzen“ initialisiert. Zum Anderen wird beim Speichern je nach aktuellen Status entweder ein neues Rechnungsdokument angelegt, das Sachkonto gespeichert oder die Zahlung vermerkt. Alle Vorgänge sind mit einer entsprechenden Statusänderung verbunden.

```
Set ELO = CreateObject("ELO.office")
ActionKey = ELO.ActionKey
```

```
Select Case ActionKey
  Case 20:
    Call PrepareNew

  Case 21:
    Call StoreNew
    Call StoreSachkonto
    Call StoreZahlung
    Call RefreshPath

End Select
```

Die Funktion PrepareNew prüft nur ab, ob das Statusfeld noch leer ist und füllt es dann mit dem initialen Wert „Bitte das Sachkonto ergänzen“.

```
Sub PrepareNew
  Dim Status

  Status = GetObjAttribByName(ELO, "RESTATUS")
  If Status = "" Then
    Status = Status_Sachkonto
    Call SetObjAttribByName(ELO, "RESTATUS", Status)
  End If
End Sub
```

Hinweis: Die Funktionen GetObjAttribByName und SetObjAttribByName wurden weiter oben bereits beschrieben und stehen im Anhang als Bibliotheksfunktion zur Verfügung.

Die Funktion StoreNew prüft nach, ob es sich um ein neues Rechnungsdokument handelt. Das wird daran erkannt, dass der Status auf Sachkonto ergänzen steht und noch kein Sachkonto eingetragen wurde. In diesem Fall wird für den Sachbearbeiter ein dynamisches Register angelegt wenn es noch nicht vorhanden war. Das wird durch die Funktion CreateDynFolder durchgeführt.

```
Sub StoreNew
  Dim Status, Path, Ids, IdList, Id, Owner
```

```
  Dim Sachkonto, DynReg

  Status = GetObjAttribByName(ELO, "RESTATUS")
  Sachkonto = GetObjAttribByName(ELO, "SKTO")
  If (Status = Status_Sachkonto) And (Sachkonto = "") Then
    Owner = GetObjAttribByName(ELO, "REOWNER")
    Path = BasePath & "Sachbearbeiter¶" & Owner
    DynReg = "!+ ,objkeys k1, objkeys k2 where " _
          & "objid = k1.parentid and " _
          & "objid = k2.parentid and " _
          & "k1.okeyname = 'REOWNER' and " _
          & "k1.okeydata = '" & Owner _
          & "' and k2.okeyname = 'RESTATUS' and " _
          & "k2.okeydata = '" & Status_Sachkonto _
          & "' and objstatus = 0 and " _
          & "objmask = " & Rechnungsmaske

    Call CreateDynFolder(Path, DynReg)
  End If
End Sub
```

Wenn ein Sachbearbeiter ein Sachkonto eingegeben hat, wird beim Speichern in der Funktion StoreSachkonto der Status entweder auf Abgelehnt oder Freigegeben umgestellt. Weiterhin werden die dynamischen Register für die Freigaben und Ablehnungen erzeugt.

```
Sub StoreSachkonto
  Dim Status, Sachkonto

  Status = GetObjAttribByName(ELO, "RESTATUS")
  Sachkonto = GetObjAttribByName(ELO, "SKTO")
  If (Status = Status_Sachkonto) And (Sachkonto <> "") Then
    If Instr(Sachkonto, "ABGELEHNT") > 0 Then
      Status = Status_Abgelehnt
    Else
      Status = Status_Freigabe
    End If
    Call SetObjAttribByName(ELO, "RESTATUS", Status)
    Call CreateAbgelehnt
    Call CreateFreigabe
  End If
End Sub
```

Wenn die Buchhaltung dann im Zustand Freigegeben ein Zahldatum einträgt, dann wird die dritte Funktion StoreZahlung aktiv und stellt den Status auf Bezahlt um.

```
Sub StoreZahlung
  Dim Status, Zahldatum

  Status = GetObjAttribByName(ELO, "RESTATUS")
  Zahldatum = GetObjAttribByName(ELO, "REZDAT")
  If (Status = Status_Freigabe) And (Zahldatum <> "") Then
    Call SetObjAttribByName(ELO, "RESTATUS", Status_Gezahlt)
  End If
End Sub
```

Die Funktionen CreateAbgelehnt und CreateFreigabe legen die dynamischen Register für die Buchhaltung an. Sie ermitteln jeweils nur den notwendigen SQL Code und rufen zur Realisierung dann die zentrale Funktion CreateDynFolder auf.

```
Sub CreateAbgelehnt
  Dim Path, Memo

  Path = BasePath & "Abgelehnte Zahlungen"
  Memo = "!+ ,objkeys k1 where objid = k1.parentid and " _
       & "k1.okeyname = 'RESTATUS' and k1.okeydata like '" _
       & Status_Abgelehnt & "' and objstatus = 0 and " _
       & "objmask = " & Rechnungsmaske

  Call CreateDynFolder(Path, Memo)
End Sub

Sub CreateFreigabe
  Dim Path, Memo

  Path = BasePath & "Freigegebene Zahlungen"
  Memo = "!+ ,objkeys k1 where objid = k1.parentid and " _
       & "k1.okeyname = 'RESTATUS' and k1.okeydata = '" _
       & Status_Freigabe & "' and objstatus = 0 and " _
       & "objmask = " & Rechnungsmaske

  Call CreateDynFolder(Path, Memo)
End Sub
```

In der Funktion CreateDynFolder wird zuerst geprüft, ob es das Register schon gibt. Nur, wenn es noch nicht vorhanden ist, wird es neu angelegt. Da diese Funktion aus einem Verschlagwortungsereignis aufgerufen wird, muss man darauf achten, dass man die aktuelle Verschlagwortung nicht zerstört. Deshalb wird vor der Neuanlage die Funktion SaveObject aufgerufen. Diese speichert den aktuellen Zustand, er wird dann später wiederhergestellt. Die Funktion CreateStructure erstellt das Register. Allerdings fehlt dann noch der SQL Code im Zusatztext, dieser wird deshalb nachträglich eingetragen. Dabei ergibt sich das Zielregister aus dem letzen Eintrag in der Rückgabe von CreateStructure (genau genommen den vorletzten, der letzte ist immer leer).

```
Sub CreateDynFolder(Path, Memo)
  Dim Ids, IdList, Id
  If ELO.LookupIndex(Path) < 1 Then
    Call ELO.SaveObject(1)
    Ids = ELO.CreateStructure(Path, 0)
    IdList = Split(Ids, "¶")
    If UBound(IdList) > 1 Then
      Id = IdList(UBound(IdList) - 1)
      If ELO.PrepareObjectEx(Id, 0, 0) > 0 Then
        ELO.ObjMemo = Memo
        Call ELO.UpdateObject
      End If
    End If
    Call ELO.SaveObject(2)
  End If
End Sub
```

Beim Speichern der Verschlagwortung wird zudem die Funktion RefreshPath aufgerufen. Diese trägt die Nummer des aktuellen Ordners in ein Cookie ein. Es gibt ein weiteres Skript als Timer Ereignis, welches diese Nummer ausliest und ein GotoId dorthin ausführt. Damit wird der Inhalt des aktuellen dynamischen Registers erneuert. Das kann nicht direkt in diesem Skript erfolgen, da es vor dem Speichern ausgeführt wird. Wenn zu diesem Zeitpunkt das dynamische Register aktualisiert würde, wäre in der Datenbank noch der alte Wert und die Anzeige würde sich nicht verändern.

```
Sub RefreshPath
```

```
  Dim Path, PathList, Id

  Path = ELO.GetTreePath(1, "¶", 200)
  PathList = Split(Path, "¶")
  Id = PathList(UBound(PathList) - 1)
  Call ELO.SetCookie("RefreshPath", "-" & Id)
End Sub
```

Asynchrone Aktualisierung der Ansicht

Zum Abschluss kommt noch das Skript welches für die Aktualisierung der dynamischen Register verantwortlich ist. Wenn ein Rechnungseintrag gespeichert wird, hinterlegt das Workflowskript in dem Cookie „RefreshPath“ die ELO Eintragsnummer des aktuellen Registers. In dem Skript TimerRefresh wird dieses Cookie intervallgesteuert ausgelesen. Wenn es einen Wert enthält, wird es gelöscht und ein GotoId auf diesen Wert ausgeführt.

```
Dim ELO, Path
Set ELO = CreateObject("ELO.office")

Path = ELO.GetCookie("RefreshPath")
If Path <> "" Then
  Call ELO.SetCookie("RefreshPath", "")
  If ELO.SelectView(0) = 1 Then
    Call ELO.GotoId(Path)
  End If
End If
```

Vollständiges Workflow Skript

Das vollständige Workflow Skript sieht dann so aus:

```
' Skript: Rechnungsworkflow
' Autor: Matthias Thiele
' Erstellt: 05.07.2010
' Letzte Änderung: 06.07.2010
'
' Dieses Programm zeigt die Erstellung
' eines einfachen Rechnungsworkflows
```

```
'
' Beim Einscannen wird die Rechnung von
' der Buchhaltung im Archiv abgelegt und
' einem Anwender zugeordnet. Dieser
' Anwender findet das Dokument in einem
' persönlichen Rechnungsordner vor und
' ergänzt das Sachkonto. Anschließend
' wird es an die Buchhaltung zur Zahlung
' zurück gegeben.


Option Explicit

Const Rechnungsmaske = 8
Const BasePath = "¶Rechnungsablage¶"
Const Status_Sachkonto = "Bitte das Sachkonto ergänzen"
Const Status_Freigabe = "Zur Zahlung freigegeben"
Const Status_Abgelehnt = "Gesperrt, keine Zuordnung möglich"
Const Status_GezahIt = "Zahlung erfolgt"

Dim ELO, ActionKey
Set ELO = CreateObject("ELO.office")
ActionKey = ELO.ActionKey

Select Case ActionKey
  Case 20:
    Call PrepareNew

  Case 21:
    Call StoreNew
    Call StoreSachkonto
    Call StoreZahlung
    Call RefreshPath

End Select

Sub PrepareNew
  Dim Status

  Status = GetObjAttribByName(ELO, "RESTATUS")
  If Status = "" Then
    Status = Status_Sachkonto
    Call SetObjAttribByName(ELO, "RESTATUS", Status)
  End If
End Sub
```

```
Sub StoreNew
  Dim Status, Path, Ids, IdList, Id, Owner
  Dim Sachkonto, DynReg

  Status = GetObjAttribByName(ELO, "RESTATUS")
  Sachkonto = GetObjAttribByName(ELO, "SKTO")
  If (Status = Status_Sachkonto) And (Sachkonto = "") Then
    Owner = GetObjAttribByName(ELO, "REOWNER")
    Path = BasePath & "Sachbearbeiter¶" & Owner
    DynReg = "!+ ,objkeys k1, objkeys k2 where " _
          & "objid = k1.parentid and " _
          & "objid = k2.parentid and " _
          & "k1.okeyname = 'REOWNER' and " _
          & "k1.okeydata = '" & Owner _
          & "' and k2.okeyname = 'RESTATUS' and " _
          & "k2.okeydata = '" & Status_Sachkonto _
          & "' and objstatus = 0 and " _
          & "objmask = " & Rechnungsmaske

    Call CreateDynFolder(Path, DynReg)
  End If
End Sub

Sub StoreSachkonto
  Dim Status, Sachkonto

  Status = GetObjAttribByName(ELO, "RESTATUS")
  Sachkonto = GetObjAttribByName(ELO, "SKTO")
  If (Status = Status_Sachkonto) And (Sachkonto <> "") Then
    If Instr(Sachkonto, "ABGELEHNT") > 0 Then
      Status = Status_Abgelehnt
    Else
      Status = Status_Freigabe
    End If
    Call SetObjAttribByName(ELO, "RESTATUS", Status)
    Call CreateAbgelehnt
    Call CreateFreigabe
  End If
End Sub

Sub StoreZahlung
  Dim Status, Zahldatum

  Status = GetObjAttribByName(ELO, "RESTATUS")
```

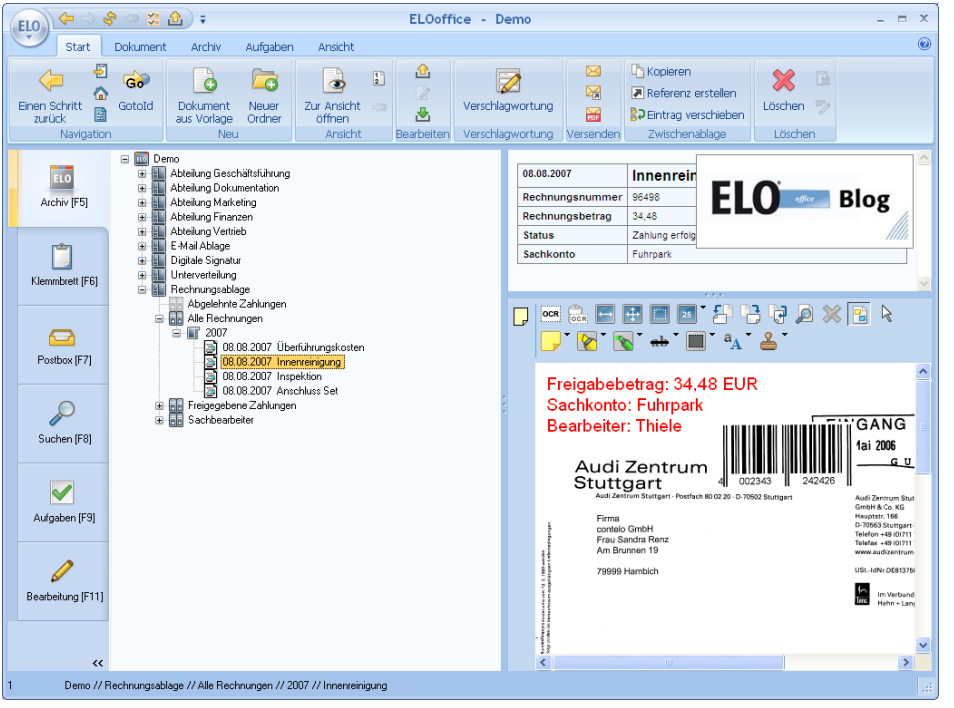
```
  ZahlDatum = GetObjAttribByName(ELO, "REZDAT")
  If (Status = Status_Freigabe) And (ZahlDatum <> "") Then
    Call SetObjAttribByName(ELO, "RESTATUS", Status_Gezahlt)
  End If
End Sub

Sub CreateAbgelehnt
  Dim Path, Memo

  Path = BasePath & "Abgelehnte Zahlungen"
  Memo = "!+ ,objkeys k1 where objid = k1.parentid and " _
       & "k1.okeyname = 'RESTATUS' and k1.okeydata like '" _
       & Status_Abgelehnt & "' and objstatus = 0 and " _
       & "objmask = " & Rechnungsmaske

  Call CreateDynFolder(Path, Memo)
End Sub

Sub CreateFreigabe
  Dim Path, Memo

  Path = BasePath & "Freigegebene Zahlungen"
  Memo = "!+ ,objkeys k1 where objid = k1.parentid and " _
       & "k1.okeyname = 'RESTATUS' and k1.okeydata = '" _
       & Status_Freigabe & "' and objstatus = 0 and " _
       & "objmask = " & Rechnungsmaske

  Call CreateDynFolder(Path, Memo)
End Sub

Sub CreateDynFolder(Path, Memo)
  Dim Ids, IdList, Id
  If ELO.LookupIndex(Path) < 1 Then
    Call ELO.SaveObject(1)
    Ids = ELO.CreateStructure(Path, 0)
    IdList = Split(Ids, "¶")
    If UBound(IdList) > 1 Then
      Id = IdList(UBound(IdList) - 1)
      If ELO.PrepareObjectEx(Id, 0, 0) > 0 Then
        ELO.ObjMemo = Memo
        Call ELO.UpdateObject
      End If
    End If
    Call ELO.SaveObject(2)
  End If
```

```
End Sub

Sub RefreshPath
  Dim Path, PathList, Id

  Path = ELO.GetTreePath(1, "¶", 200)
  'MsgBox Path

  PathList = Split(Path, "¶")
  Id = PathList(UBound(PathList) - 1)
  Call ELO.SetCookie("RefreshPath", "-" & Id)
End Sub


' ##########################################
' Die folgenden Routinen sind Library Routinen
' und sollten nicht verändert werden


' Schreibt den Wert "AttribValue" in die
' Indexzeile mit dem Gruppennamen "AttribName".
' Der erste Parameter enthält das ELO Objekt.
Sub SetObjAttribByName(ELO, AttribName, AttribValue)
  Dim i, LocalName

  LocalName = LCase(AttribName)
  For i = 0 to 49
    If LCase(ELO.GetObjAttribKey(i)) = LocalName Then
      Call ELO.SetObjAttrib(i, AttribValue)
      Exit For
    End If
  Next
End Sub

' Liest den aktuellen Wert der Indexzeile mit
' dem Namen "AttribName". Der erste Parameter
' enthält das ELO Objekt
Function GetObjAttribByName(ELO, AttribName)
  Dim i, LocalName

  GetObjAttribByName = ""
  LocalName = LCase(AttribName)
  For i = 0 to 49
    If LCase(ELO.GetObjAttribKey(i)) = LocalName Then
      GetObjAttribByName = ELO.GetObjAttrib(i)
```

```
    Exit For
  End If
Next
End Function
```

One-Click Zahlungsquittung

Als Ergänzung zu dem oben aufgeführten Skript kann man der Buchhaltung mit dieser Erweiterung das Leben noch etwas vereinfachen. Wenn eine Rechnung freigegen und bezahlt wurde, muss in der Buchhaltung noch mal die Verschlagwortung geöffnet werden, das aktuelle Datum in der Zahldatum Indexzeile eingetragen werden und wieder gespeichert werden. Damit wird der Status auf „Bezahlt“ gesetzt. Diese Funktion kann man auch auf eine Schaltfläche legen, die mit einem Skript belegt ist, welches diese Aktionen ausführt. Zusätzlich legt dieses noch einen Textstempel auf die Tiff-Datei mit den wichtigsten Rechnungsinformationen.



Der Textstempel wird über die Funktion AddNoteEx2 erzeugt

```
Const StampPrefix = "9+240000FF0360Arial                    "

Text = StampPrefix & "Freigabebetrag: " _
       & GetObjAttribByName(ELO, "RE-Betrag") _
       & " EUR" & vbCrLf & "Sachkonto: " _
       & GetObjAttribByName(ELO, "SKTO") _
       & vbCrLf & "Bearbeiter: " _
       & GetObjAttribByName(ELO, "REOWNER")

Call ELO.AddNoteEx2(Id, 23, Text, 1, 50, 50, 0, 1400, 200, "")
```

Der Text für den Stempel besteht aus einem Präfix welches Informationen zur Textausrichtung, Farbe, Font und Zeichensatzgröße enthält. Dieser Teil hat eine feste Formatierung, auch die Zahl der Leerzeichen am Ende ist wichtig. Wenn man hier Fehler macht, kann das später zu erheblichen Fehlfunktionen im ELO bis zu einer Schutzverletzung führen. Danach folgt ein beliebiger Text, in diesem Beispiel werden die Verschlagwortungsinformationen aus dem Rechnungsbetrag, dem Sachkonto und dem Bearbeiter übernommen.

Bei den Indexwerten passieren im Wesentlichen nur bekannte Dinge. Auffällig ist lediglich die Ermittlung des aktuellen Datums. In der Ablagemaskendefinition wurde das Zahldatumsfeld als ISO Datum angelegt. Aus der Sicht des Anwenders macht das keinen Unterschied, er bekommt ein normales Datum angezeigt. Der Skriptentwickler muss darauf aber Rücksicht nehmen – er muss ein ISO Datum (z.B. 20101231 für den 31.12. 2010) anliefern.

```
Sub UpdateIndex
  Dim PayDate

  PayDate = Date
  PayDate = Right(PayDate, 4) & Mid(PayDate, 4, 2) & Left(PayDate, 2)
  Call SetObjAttribByName(ELO, "REZDAT", PayDate)
  Call SetObjAttribByName(ELO, "RESTATUS", Status_Gezahlt)
  Call ELO.UpdateObject
End Sub
```

Zum Abschluss muss die Ansicht aktualisiert werden. Dass passiert über den gleichen Mechanismus wie beim Skript Rechnungsworkflow. Es wird ein Cookie gesetzt welches über ein asynchrones Timer Skript die Aktualisierung auslöst.

Das komplette Skript sieht dann so aus:

```
' Skript: Rechnungsworkflow
' Autor: Matthias Thiele
' Erstellt: 06.07.2010
' Letzte Änderung: 06.07.2010
'
' Dieses Programm dient zur Ergänzung
' des Skripts Rechnungsworkflow. In diesem
' Workflow wird in einem letzten Schritt
' das Zahlungsdatum eingetragen und der
' Status auf Bezahlt gesetzt. Diese Schritte
' können durch dieses Skript mit einem Klick
' durchgeführt werden, wenn das Skript auf
' eine Schaltfläche gelegt wird.
'
' Zusätzlich fügt dieses Skript einen
' Textstempel mit den aktuellen
' Indexinformationen ein.

Option Explicit

Const StampPrefix = "9+240000FF0360Arial                    "
Const Status_Gezahlt = "Zahlung erfolgt"
Const MinVersion = 900038000

Dim ELO, Id
Set ELO = CreateObject("ELO.office")

If ELO.Version < MinVersion Then
  Call ELO.MsgBox("Dieses Skript benötigt eine neuere ELO Version" _
              & vbCrLf & "Aktuelle Version: " & ELO.Version _
              & vbCrLf & "Benötigte Version: " & MinVersion, _
              "ELO", vbOkOnly)
Else
  Id = ELO.GetEntryId(-1)
```

```
  If Id < 2 Then
    Call ELO.MsgBox("Sie müssen zuerst eine Rechnung zur " _
                & "Zahlungsquittierung auswählen.", _
                  "ELO", vbOkOnly)
  Else
    ProcessEntry(Id)
  End If
End If


Sub ProcessEntry(Id)
  If ELO.PrepareObjectEx(Id, 0, 0) Then
    Call UpdateIndex
    Call AddNote(Id)
    Call RefreshPath
  Else
    Call ELO.MsgBox("Der Eintrag konnte nicht aus der " _
                & "Datenbank gelesen werden.", _
                "ELO", vbOkOnly)
  End If
End Sub

Sub UpdateIndex
  Dim PayDate

  PayDate = Date
  PayDate = Right(PayDate, 4) & Mid(PayDate, 4, 2) & Left(PayDate, 2)
  Call SetObjAttribByName(ELO, "REZDAT", PayDate)
  Call SetObjAttribByName(ELO, "RESTATUS", Status_Gezahlt)
  Call ELO.UpdateObject
End Sub

Sub AddNote(Id)
  Dim Text

  Text = StampPrefix & "Freigabebetrag: " _
       & GetObjAttribByName(ELO, "RE-Betrag") _
       & " EUR" & vbCrLf & "Sachkonto: " _
       & GetObjAttribByName(ELO, "SKTO") _
       & vbCrLf & "Bearbeiter: " _
       & GetObjAttribByName(ELO, "REOWNER")

  Call ELO.AddNoteEx2(Id, 23, Text, 1, 50, 50, 0, 1400, 200, "")
End Sub
```

```
Sub RefreshPath
  Dim Path, PathList, Id

  Path = ELO.GetTreePath(1, "¶", 200)
  PathList = Split(Path, "¶")
  Id = PathList(UBound(PathList) - 1)
  Call ELO.SetCookie("RefreshPath", "-" & Id)
End Sub

' ##########################################
' Die folgenden Routinen sind Library Routinen
' und sollten nicht verändert werden


' Schreibt den Wert "AttribValue" in die
' Indexzeile mit dem Gruppennamen "AttribName".
' Der erste Parameter enthält das ELO Objekt.
Sub SetObjAttribByName(ELO, AttribName, AttribValue)
  Dim i, LocalName

  LocalName = LCase(AttribName)
  For i = 0 to 49
    If LCase(ELO.GetObjAttribKey(i)) = LocalName Then
      Call ELO.SetObjAttrib(i, AttribValue)
      Exit For
    End If
  Next
End Sub

' Liest den aktuellen Wert der Indexzeile mit
' dem Namen "AttribName". Der erste Parameter
' enthält das ELO Objekt
Function GetObjAttribByName(ELO, AttribName)
  Dim i, LocalName

  GetObjAttribByName = ""
  LocalName = LCase(AttribName)
  For i = 0 to 49
    If LCase(ELO.GetObjAttribKey(i)) = LocalName Then
      GetObjAttribByName = ELO.GetObjAttrib(i)
      Exit For
    End If
  Next
End Function
```

Thematisch sortierte Liste der OLE Schnittstellenaufrufe

Eine vollständige Liste aller OLE Automation Befehle finden Sie in dem Download Dokument „OLE Automation“. Zum Nachschlagen sollten Sie auf jeden Fall dieses verwenden. Allerdings sind dort die Befehle alphabetisch aufgelistet, eine Gruppenzugehörigkeit wird nur schwer erkennbar. Deshalb werden im Folgenden die wichtigsten OLE Befehle gruppiert aufgeführt.

Allgemeine Systemaufrufe

ActionKey	Dieses Property kann innerhalb von Ereignisaufrufen gelesen werden. Hier wird der aktuelle Systemzustand anhand einer Nummer beschrieben (z.B. Verschlagwortung betreten: ActionKey = 20).
ActiveUserId	Interne ELO Nummer des aktuell angemeldeten Anwenders.
CheckUpdate	Wenn umfangreiche Änderungen am Archiv per OLE Schnittstelle vorgenommen werden, kann die ständige Aktualisierung der Oberfläche sehr viel Zeit in Anspruch nehmen. Über diesen Befehl können Sie die Aktualisierung vorher ab- und nachher wieder anschalten.
ClickOn	Simuliert einen Anwender-Klick auf einen Hauptmenüeintrag oder eine Schaltfläche. Kontextmenüfunktionen können nicht erreicht werden.
DateToInt	ELO verwendet intern ein numerisches Datumsformat (Anzahl der Minuten seit dem 1.1.1900). Dieser Befehl wandelt ein Datumstext in das numerische Format um.
DebugOut	Gibt einen beliebigen Text zur Protokollierung in der Reportdatei und dem ELO Debug Fenster aus.

EloWindow	Minimiert oder Maximiert das ELO Anzeigefenster.
FreezeDoc	Konvertiert ein Dokument in das Tiff-Format.
FromClipboard	Liest einen Text aus dem Windows Clipboard aus.
GetArcName	Liest den Namen des aktuell aktiven Archivs aus.
GetCookie	Skripte können mittels Get/Set-Cookie Daten zwischen den unterschiedlichen Aufrufen im ELO Client zwischenspeichern und wieder auslesen.
GetDocFromObj	Ermittelt zu einer ELO Objektnummer die interne Dateinummer des Dokuments oder der Dateianbindung.
GetLastDocId	Gibt die größte Dokumentendateinummer im Archiv zurück.
GetPopupObjectId	Wenn ein Ereignisskript über ein Kontextmenü aufgerufen wurde, kann man über diesen Befehl die Objektnummer abfragen, zu der dieser Aufruf gehört.
IntToDate	Wandelt den numerischen Eintrag eines internen ELO Datumswertes in ein Datum im Textformat um.
LookupDelimiter	Über diese Einstellung können Sie das Trennsymbol für den LookupIndex Befehl abfragen und setzen. Im Normalfall sollte man jedoch die Voreinstellung auf das Pilcrow Symbol ¶ nicht verändern.
OfficeMaskNo	Gibt die konfigurierte Dokumentenmaskennummer für neue Office Dokumente zurück.
RunEloScript	Diese Funktion ruft aus einem Skript heraus die Ausführung eines anderen Skripts auf.

SaveObject	Die OLE Automation Schnittstelle bietet nur ein Verschlagwortungsobjekt an. Das führt zu Problemen, wenn Sie in einer Ereignisroutine neue Einträge anlegen wollen, da diese Daten die Einstellungen des gerade aktiven Eintrags überschreiben. In diesem Fall kann der aktuelle Zustand mittels SaveObject gesichert und nach der Erstellung des neuen Eintrags wiederhergestellt werden. Diese Funktion darf nicht rekursiv verwendet werden, da der Sicherungsspeicher auch nur ein Objekt tief ist.
ScriptActionKey	Über dieses Property werden Informationen vom laufenden Script an ELO zurückgeliefert.
SelectLine	Über die Funktion SelectLine können Sie einen Eintrag der aktuellen Auswahlliste selektieren. Dieser Befehl kann in der Archivansicht, dem Klemmbrett, der Postbox, der Trefferliste und in der Aufgabenliste verwendet werden.
SelectView	Über die Funktion SelectView können Sie bestimmen welche Arbeitsoberfläche sichtbar sein soll. Weiterhin können Sie damit abfragen, welche gerade sichbar ist.
Sleep	Diese Funktion führt eine Pause mit einer einstellbaren Anzahl von Millisekunden aus. Dabei können Sie zusätzlich vor und/oder nach der Wartezeit ein ProcessMessages ausführen. Dieser Aufruf führt dazu, dass der Client aktiv bedienbar bleibt.
Status	Diese Funktion schreibt im aktuellen ELO Hauptfenster einen Text in die Statuszeile.

TextParam	Das Property TextParam enthält unterschiedliche Parameter, die von einem Script-Event an das Script oder von dem Script zurück an das Event übergeben werden sollen. Der Inhalt dieses Properties wird beim jeweiligen Script Event beschrieben. Zu allen anderen Zeitpunkten hat dieses Property einen zufälligen Wert und sollte auch nicht verändert werden.
ToClipboard	Diese Funktion übergibt einen Text an das Windows Clipboard
UnselectLine	Diese Funktion deselektiert einen Eintrag in der aktuellen Ansicht. Diese Funktion darf nur in Ansichten angewendet werden, in welchen Mehrfachselektionen (z.B. Klemmbrett) zulässig sind. In den anderen Ansichten (z.B. Archivansicht) kann es zu zufälligen Ergebnissen kommen.
Version	Über dieses Property kann die ELO Versionsnummer abgefragt werden.
WindowState	Über dieses Property kann der Status des ELOoffice Fensters abgefragt werden. 1. Normal 2. Minimiert 3. Maximiert

Verschlagwortung bearbeiten

Basisdaten aus der Verschlagwortung: diese Werte befinden sich im Verschlagwortungsdialog auf der Basis-, Zusatztext- und der Optionenseite.

ObjShort	Kurzbezeichnung – Dieser Wert muss zwingend gesetzt werden. Wenn ein Objekt ohne Kurzbezeichnung gespeichert wird, kann es im Archiv nicht richtig angezeigt werden und führt an einigen Stellen zu Problemen bei der Verarbeitung.
Obj**I**Date	Ablagedatum (**I**nternes Datum) – Das Ablagedatum wird beim Speichern automatisch ermittelt und muss nicht gesetzt werden.
Obj**X**Date	Dokumentendatum (e**X**ternes Datum) – Das Dokumentendatum kann optional gesetzt werden. Anders als der Name vermuten lässt, darf es auch für Ordner gesetzt werden. Beachten Sie unbedingt, dass dieses Datum nur Tagesgenau (z.B. auf den 13.04.2010) und nicht Minutengenau (13.04.2010 14:20) gesetzt werden darf.
ObjSReg	Versionsnummer – Wenn die Anzeige der Versionsinformation im Verschlagwortungsdialog aktiviert ist, wird dieser Wert angezeigt. Er wird aber bei jedem CheckIn Vorgang von der neuen Versionsnummer überschrieben.
ObjMemo ObjMemoInfo	Zusatztext – der Zusatztext besteht aus zwei Teilen. Den sichtbaren Teil können Sie mit ObjMemo setzen. Er kann vom Anwender im Verschlagwortungsdialog eingesehen werden und verändert werden. Der zweite, unsichtbare Teil wird mit ObjMemoInfo gesetzt. Er kann nur per Skript gelesen oder geschrieben werden, für den Anwender ist er weder sichtbar noch änderbar. Beide Teile zusammen können maximal 30000 Zeichen enthalten.
ObjTypeEx	Objekttyp – bei Ordnern „Schrank – Ordner – Register …“ (Wert 1 bis 32) und bei Dokumenten „Dokument – Word – Excel …“ (Wert 254 bis 285).

ObjOwner	Eigentümer des Eintrags – dieser Wert ist für den Anwender im Feld „Abgelegt von“ sichtbar. In der OLE Schnittstelle wird die ELO Anwendernummer gelesen und geschrieben, nicht der Name.
ObjAcl	Zugriffsberechtigungen auf diesen Eintrag
DocKind	Marker – Farbmarkierung für Ordner oder Dokumente. Dieser Wert enthält keine direkte Farbe sondern eine Nummer als Index in die Liste der definierten Farbmarker.
ObjFlags	Verschiedene Einstellungen, z.B. Sortierung, Versionskontrolle, Verschlüsselung, Volltextkennzeichen
ObjVDate	Verfallsdatum – Verfallsdatum des Dokuments.

Zugriff auf die Werte der Indexzeilen

GetObjAttrib	Liest den Wert einer Indexzeile aus. Die Nummer der Indexzeile wird als Parameter übergeben, die Nummerierung beginnt mit 0 für die erste Zeile.
SetObjAttrib	Setzt den Wert einer Indexzeile. Beachten Sie bitte, dass die OLE Schnittstelle keine Typprüfung vornimmt. Wenn eine Indexzeile als ISO-Datumsfeld definiert wurde, können hier trotzdem Texte eingetragen werden. Diese führen dann später aber zu einer unsinnigen Anzeige im Verschlagwortungsdialog.
GetObjAttribMax GetObjAttribMin	Gibt die maximale und minimale Eingabelänge zu einer Indexzeile aus. Diese Information können Sie zur Prüfung des Textes vor einem SetObjAttrib Aufruf verwenden. Die OLE Schnittstelle prüft diese Werte nicht ab.

GetObjAttribType	Gibt den Typ der Indexzeile zurück. Diese Information können Sie zur Prüfung der Formatierung und erlaubten Werte eines Textes vor einem SetObjAttrib Aufruf verwenden. Die OLE Schnittstelle prüft diese Werte nicht ab.
GetObjAttribKey	Gibt den Gruppennamen einer Indexzeile zurück. Falls Sie mit einem Skript bestimmte Werte (z.B. die Kundennummer KDNR) in unterschiedlichen Masken füllen müssen und sich diese in unterschiedlichen Indexzeilen befinden, können Sie diese auch über den GetObjAttribKey Wert suchen und ermitteln.
AttId DocId	Interne ELO Datei Nummer der aktuellen Arbeitsversion der Dateianbindung oder des Dokuments. Dieser Wert darf nur auf einen gültigen Wert aus der Liste aller Versionen dieses Eintrags umgestellt werden. Wenn hier ein ungültiger Wert eingetragen wird, kann es zu erheblichen Fehlfunktionen im Programmablauf kommen.

Allgemeine Funktionen zur Verschlagwortung

CheckObjAcl	Prüft die Zugriffsberechtigungen auf das angegebene Objekt. Wenn eine Objektnummer 0 angegeben wird, prüft diese Funktion das aktuelle Objekt (ObjAcl).
DoEditObjectEx	Ruft den Verschlagwortungsdialog zum aktuell aktiven Verschlagwortungseintrag auf. Dieser kann in einem Ereignisaufruf bereits vorhanden sein oder mittels PrepareObjectEx gelesen oder erzeugt werden.

EditDlgActive	Gibt an, ob bereits ein Verschlagwortungsdialog aktiv ist. Dieser Befehl sollte z.B. in Ereignisroutinen aufgerufen werden bevor ein Verschlagwortungsdialog per Skript geöffnet wird, wenn das Ereignis möglicherweise aus der Verschlagwortung heraus gestartet wurde (Ein Verschlagwortungsdialog im Verschlagwortungsdialog ist nicht möglich).
GetEntryName	Liefert die Kurzbezeichnung zu einer ELO Objektnummer zurück.
GetGuidFromObj	Liefert die interne GUID (eindeutige Objektkennzeichnung) zu einer ELO Objektnummer zurück.
GetIndexGroups	Liefert zu einer Verschlagwortungszeile eine Liste aller bereits eingetragenen Werte in diesem Archiv. Diese Funktion sollte nur zu Indexzeilen aufgerufen werden, die eine überschaubare Anzahl unterschiedlicher Werte besitzen.
LockObject	Sperrt oder entsperrt einen ELO Eintrag für die weitere Bearbeitung durch andere Anwender.
OkEnabled	Mit dem Property OkEnabled kann geprüft werden, ob der Ok-Button im ELO Verschlagwortungsdialog aktiviert ist und der Anwender Änderungen an den Indexdaten vornehmen darf. Das Property muss abgeprüft werden, wenn der ELO-eigene Verschlagwortungsdialog durch einen eigenen Dialog (z.B. eine Visual Basic Maske) ersetzt wird. Nur wenn der der Rückgabewert den Wert 1 besitzt, kann durch Setzen des Properties ScriptActionKey auf den Wert 10 ein Ok-Click an ELO durchgemeldet werden.

PrepareObjectEx	Über diese Funktion können Sie einen neuen Dokumenten- oder Ordnereintrag vorbereiten oder einen bestehenden Eintrag lesen. Dieser Eintrag kann dann über die Properties, wie z.B. ObjShort oder ObjXDate bearbeitet werden. Nach Abschluss dieser Aktionen kann der Datensatz mit UpdateObject in die Datenbank zurückgeschrieben werden.
PrintDocument	Diese Funktion druckt das gerade angezeigte Dokument.
UpdateObject	Über die Funktion UpdateObject fügen Sie einen Eintrag in die Datenbank ein. Dieses Objekt muss vorher mittels PrepareObjectEx erzeugt bzw. aus der Datenbank gelesen worden sein.

Weitere Properties mit internen Daten

ObjGuid	Eindeutige archivübergreifende Kennzeichnung – Im Gegensatz zur Objektnummer, die sich Export/ Importvorgänge ändern kann, bleibt diese Nummer im Normalfall immer gleich. Für die Integration/ Verlinkung mit anderen Programmen ist diese deshalb vorzuziehen. Die GUID ist keine einfache Zahl, sondern ein ca. 40 stelliger Text mit einer Windows GUID. Mit dem Befehl GetObjFromGuid kann eine GUID bei Bedarf in eine normale ELO Objektnummer übersetzt werden.

ObjMainParent	Vorgängerordner des aktuellen Elements. Im Normalfall sollte man nur lesend zugreifen. Wichtig: Wenn Sie ein Dokument in einen anderen Ordner verschieben wollen, dann reicht es nicht aus, diesen Wert einfach nur zu ändern. Für diese Aktion benötigen Sie den Befehl InsertRef.
ObjStatus	Löschstatus – Gibt an, ob ein Eintrag als gelöscht markiert wurde. 0 = normaler Zustand, -1 = gelöscht markiert.

Behandlung der Dokumentendateien und Versionen

GetDocRefComment	Gibt die Versionsnummer und den Versionskommentar zu einem Dokument zurück.
GetDocumentExt	Gibt den Dateityp (Extension) zu einem Dokument zurück.
GetDocumentPath	Liest aus einem Dokument die Dokument oder Attachment Datei und gibt einen Zugriffspfad auf die Datei zurück. Normale Archivdokumente sind schreibgeschützt, der Anwender kann deshalb über den Status eine freie Kopie anfordern (AUTO_WRITEACCESS). Diese Kopie ist allerdings nur begrenzte Zeit gültig und wird beim beenden von ELO automatisch gelöscht.

GetDocumentPathVersion	Liest aus einem ELO Dokument die Dokumenten- oder Attachmentdatei und gibt einen Zugriffspfad auf die Datei zurück. Bei versionskontrollierten Dokumenten kann über den Parameter *Version* auf vorherige Versionen zugegriffen werden. Dabei wird *Version* mit 0 beginnend (=aktuelle Version) so lange um 1 erhöht, bis ein leerer String zurückgeliefert wird.
GetDocumentSize	Liefert die Größe der Dokumentendatei zu einem ELO Dokument zurück.
GetHistDoc	Liefert eine Dateinummer aus der Trefferliste des letzten LookupHistMD5Ext zurück.
GetHistObj	Liefert eine Objektnummer aus der Trefferliste des letzten LookupHistMD5Ext zurück.
GetMD5Hash	Über diese Funktion können Sie den MD5 Hash zu einer Datei oder zu einem Datenblock ermitteln. Wenn Sie als Parameter einen Dateinamen angeben, erhalten Sie den Hash zu der Datei. Wenn Sie einen String, beginnend mit den Zeichen „##“ übergeben, erhalten Sie den Hash Wert zu diesem String (die ##-Zeichen werden dabei nicht mitgezählt).
GetObjFromDocEx	Ermittelt die ELO Objektnummer zu einer gegebenen ELO Dateinummer.
GetObjFromGuid	Ermittelt zu einer GUID die interne ELO Objektnummer.
InsertDocAttachmentEx	Mithilfe dieser Funktion können Sie an ein bestehendes ELO Dokument eine Dateianbindung anfügen.

LookupHistMD5Ext	Diese Funktion ermittelt die Anzahl von Dokumenten, die einen vorgegebenen MD5 Hash Wert besitzen.
UpdateDocumentEx	Diese Funktion fügt an ein ELO Dokument eine neue Dokumentendatei an. Je nach Status des Dokuments wird die alte Version überschrieben (freie Bearbeitung), eine neue Version angelegt (Versionskontrolliert) oder die Funktion zurückgewiesen (Revisionssicher).
ViewFileName	Das Property ViewFileName enthält den Namen des im Dokumentenviewer angezeigten Dokuments.

Barcodebehandlung

GetBarcode	Liest einen Wert aus der Liste der erkannten Barcodes aus. Diese wurden vorher mittels ReadBarcodes ermittelt.
ObjBarcodeInfo	Über das Property ObjBarcodeInfo können Sie Barcode Konfiguration für den ausgewählten Dokumententyp abfragen. Dieser Text wird über die Ablagemaskenverwaltung im Feld BarcodeInfo eingetragen und der internen Barcodeanalyse zur Verfügung gestellt. Sie können diesen Text aber auch extern auswerten, z.B. um eigene Barcode Komponenten einzufügen.

ReadBarcodes	Diese Funktion liest Barcodes, die sich auf einem Postbox-TIFF-Dokument befinden. Der Name der Postboxdatei kann über SourceFile übergeben werden. Alternativ hierzu kann in diesem Parameter eine Zeilennummer (#0, #1, #2 ...) übergeben werden. Als Dateiname wird dann die entsprechende Datei aus der Postliste verwendet. Die Funktion liefert die Anzahl der erkannten Barcodes zurück, die Barcodes können mit Hilfe der Funktion *GetBarcode* abgerufen werden.

Randnotizen und Haftnotizen

DeleteNote	Löscht eine Rand-/ Haftnotiz. Diese muss zuvor mittels FindFirstNote oder FindNextNote aktiviert worden sein.
FindFirstNote	Aktiviert die erste Rand-/ Haftnotiz des aktiven Dokuments. Dieses kann über PrepareObjectEx aktiviert werden.
FindNextNote	Aktiviert die nächste Rand-/ Haftnotiz des aktivenDokuments.
NoteText	Über dieses Property kann der Text einer Randnotiz gesetzt werden. Das ist auch für Haftnotizen möglich, hier muss aber unbedingt darauf geachtet werden, dass der Vorspann im Text mit der Information über den Zeichensatz und die Zeichengröße nicht beschädigt werden darf.
NoteType	Setzt den Typ der Rand-/ Haftnotiz.
UpdateNote	Die Funktion aktualisiert eine Haftnotiz in der Datenbank, die zuvor mit FindFirstNote oder FindNextNote eingelesen wurde.

Ablagestruktur und Listen

CollectChildList	Gibt zu einem Ordner die Liste aller Untereinträge in einer Textdarstellung zurück. Die Objektnummern sind jeweils durch einen Doppelpunkt getrennt.
CollectLinks	Gibt zu einem ELO Objekt eine Komma-Liste aller damit per Link verbundenen Dokumente zurück.
CreateStructure	Erzeugt einen neuen Ordnerpfad falls er im Archiv noch nicht vorhanden ist.
DeleteObj	Löscht den angegebenen ELO Eintrag. Falls es sich bei diesem um einen Ordner handelt, werden auch alle Untereinträge gelöscht.
DoSelArcTree	Zeigt einen Dialog mit Archivbaum zur Auswahl eines Eintrags an.
GetEntryId	Diese Funktion liefert die interne ELO Objektnummer einer Zeile der Archiv-, Postbox- oder Suchansicht zurück. Über diesen Weg können Sie eine Liste aller sichtbaren Objekte erhalten indem Sie beginnend mit Zeile 0 aufsteigend solange die Funktion GetEntryId aufrufen, bis Sie eine 0 zurückerhalten. Wenn Sie den Aufruf in der Wiedervorlage starten, erhalten Sie statt der Objektnummer eine Wiedervorlagenummer zurück. Mittels GetEntryId(-1) kann der aktuelle selektierte Eintrag ermittelt werden und über GetEntryId(-2) der zuletzt gespeicherte Eintrag.
GetListEntry	Gibt einen Eintrag aus der internen, nicht sichtbaren Trefferliste der Aufrufe CollectWv und DoInvisibleSearch zurück.
GetTreePath	Liefert den ausgewählten Pfad in der Archivansicht auf den aktuellen Eintrag zurück.

GotoId	Springt in der Archivansicht zum Eintrag mit der angegebenen Objektnummer. Falls es mehrere Wege über Referenzen gibt, wird der Hauptpfad verwendet.
GotoPath	Öffnet in der Archivansicht den angegebenen Pfad. Dieser Pfad kann auch über referenzierte Ordner laufen.
InsertRef	Der Befehl InsertRef wird sowohl für das Verschieben wie auch für das Referenzieren von Dokumenten verwendet. Wenn der „OldParent“ Parameter -1 enthält, wird eine neue Referenz erzeugt. Steht der Parameter auf der Eintragsnummer der alten Postion (diese kann über ObjMainParent abgefragt werden), dann wird das Objekt an die neue Position verschoben.
LookupIndex	Ermittelt die interne ELO ObjektId über einen Zugriffspfad. Hierzu übergeben Sie einen ObjektIndex auf den gesuchten Eintrag und Sie erhalten die zugehörende ObjektId zurück.
RemoveRef	Mittels dieser Funktion können Referenzen auf zusätzliche Ordner gelöscht werden.
SelectArcListLine	Diese Funktion selektiert einen Eintrag in der Suchansicht. Die Zeilennummer läuft dabei von 0 bis Anzahl der Einträge minus 1.
TreeWalk	Diese Funktion läuft vom Startknoten über alle Unterknoten und ruft für jeden Eintrag ein Skript auf. Hiermit können Sie also ganze Strukturen durch einen Aufruf bearbeiten lassen.
UnselectArcListLine	Diese Funktion deselektiert einen Eintrag in der rechten Liste der Archivansicht.

Suchen und Finden

DoFullTextSearch	Führt eine Volltextsuche aus. Die Trefferliste steht anschließend in der Suchansicht zur Verfügung. Einzelne Dokumente aus der Liste können mittels SelectLine zur Ansicht gebracht werden.
DoInvisibleSearch	Führt eine Suche aus ohne die Trefferliste zu verändern. Die gefundenen Einträge können wie die normalen Treffer mittels GetEntryId abgefragt werden, allerdings beginnt die unsichtbare Trefferliste mit der Zeilennummer 268435456 (das ist 0x10000000).
DoSearchEx	Führt eine Suche im Archiv aus und zeigt die Trefferliste in der Suchansicht an.
SearchListColumns	Mit diesem Property können Sie die Spaltenüberschriften der Trefferliste ermitteln. Die einzelnen Überschriften sind durch ein „|“-Zeichen getrennt. Der String kann mit Hilfe der VBScript-Funktion „Split“ in einzelne Strings zerlegt werden.
SearchListLineId	Ermittelt die ELO Objektnummer einer Zeile aus der Trefferliste.
SearchListLineSelected	Ermittelt, ob eine Zeile in der Trefferliste selektiert ist.
SelectSearchListLine	Diese Funktion selektiert einen Eintrag in der Suchansicht. Die Zeilennummer läuft dabei von 0 bis Anzahl der Einträge minus 1.
SortSearchList	Mit dieser Funktion kann die Trefferliste in der Suchansicht des ELO Clients nach einer bestimmten Spalte sortiert werden. In den Suchoptionen von ELO muss hierzu die mehrspaltige Anzeige aktiviert sein.

UnselectSearchListLine	Diese Funktion deselektiert einen Eintrag in der Suchansicht.

Postboxinhalt bearbeiten

ActivePostFile	Gibt den Namen der aktuell aktiven Postboxdatei zurück.
GetPostDir	Liefert den Verzeichnispfad auf das Postboxverzeichnis des aktuellen Anwenders zurück.
LoadPostImg	Diese Funktion lädt eine Datei in den Viewer der aktuellen Postboxansicht. Das Dokument muss sich dabei nicht in der ELO Postbox befinden.
MovePostboxFile2	Kopiert oder verschiebt eine Datei aus der Postbox in die Postbox eines anderen Anwenders.
MoveToArchiveEx	Überträgt eine Postboxdatei in das Archiv.

OrientFile	Mit dieser Funktion können Sie eine Bilddatei in der Postbox analysieren lassen, eine eventuelle Rotation um 90, 180 oder 270 Grad wird korrigiert. Die Analyse erfolgt unter Verwendung des OCR-Systems. Vor Verwendung der Funktion muss die Funktion OCRInit aufgerufen werden, nach Beendigung die Funktion OCRExit. Die Funktion arbeitet auch mit Multipage TIFF-Dateien. Als Dateiname kann ein Eintrag aus der Postbox übergeben werden (ohne Pfad, nur der Dateiname) oder ein Index in die Postliste (durch ein # gekennzeichnet, z.B. #0 ist der erste Eintrag in der Postliste).
PostBoxLineSelected	Mit dieser Funktion kann getestet werden, ob eine Zeile in der Postbox selektiert ist.
RotateFile	Mit dieser Funktion können Sie eine Bilddatei in der Postbox um 90, 180 oder 270 Grad drehen. In der aktuellen Version lassen sich nur Einseiten-Tiffs drehen, mehrseitige Dokumente können nicht bearbeitet werden. Als Dateiname kann ein Eintrag aus der Postbox übergeben werden (ohne Pfad, nur der Dateiname), ein Index in die Postliste (durch ein # gekennzeichnet, z.B. #0 ist der erste Eintrag in der Postliste) oder auch ein Leerstring. In diesem Fall wird die aktuelle Postboxdatei (ActivePostFile, z.B. vom vorhergehenden Scanvorgang) verwendet.
SelectAllPostBoxLines	Diese Funktion selektiert alle Zeilen der Postbox.

SelectPostboxLineEx	Diese Funktion selektiert einen Eintrag in der Postbox. Optional kann dabei die Dokumentendatei neu in den Viewer geladen werden.
StoreDirect	Diese Funktion legt die in der Postbox selektierten Einträge im Archiv ab, und entspricht damit dem Menüpunkt ‚Direktablage ins Archiv' im Kontextmenü der Postbox. Vor dem Aufruf muß in der Archivansicht das gewünschte Register geöffnet werden.
StoreKeyword	Diese Funktion legt die in der Postbox selektierten Einträge per Schlagwortablage im Archiv ab, sie entspricht dem Menüpunkt ‚Schlagwortablage ins Archiv' im Kontextmenü der Postbox.
StoreMulti	Diese Funktion legt die in der Postbox selektierten Einträge im Archiv ab, sie entspricht dem Menüpunkt ‚Anonyme Registerablage ' im Kontextmenü der Postbox.
UnselectAllPostboxLines	Diese Funktion deselektiert alle Zeilen der Postbox.
UnselectPostboxLine	Diese Funktion deselektiert einen Eintrag in der Postbox.
UpdatePostboxEx	Die Funktion UpdatePostbox löst ein erneutes Sammeln des Postbox-Inhaltes aus. Neben den reinen Postbox-Dateien werden noch die Tiff-Printer Dateien, Netzwerkscanner-Dateien und Outlook-Einträge übernommen.

Allgemeine Dateibefehle

CheckFile	Mittels CheckFile können Sie prüfen, ob ein exklusiver Zugriff auf eine Datei möglich ist (OptionNo = 0). Damit kann man z.B. vor einem CheckIn Vorgang prüfen, ob das Dokument noch in Word geöffnet ist.
CheckFileHash	Ermittelt den MD5 Hash Wert einer Datei.
DoExecuteEx	Über diesen Befehl kann ein anderes Programm aufgerufen werden oder ein Dokument zur Bearbeitung aktiviert werden.
GetDocExt	Gibt zu einer ELO Dokumentendatei die Dateikennung (Extension) zurück.
GetDocumentOrientation	Mit dieser Funktion können Sie eine Bilddatei in der Postbox analysieren lassen, um eine eventuelle Rotation um 90, 180 oder 270 Grad zu erkennen. Die Analyse erfolgt unter Verwendung des OCR-Systems. Vor Verwendung der Funktion muss die Funktion OCRInit aufgerufen werden, nach Beendigung die Funktion OCRExit.
LookupDocType	Ermittelt aus der Dateikennung (Extension) den voreingestellten ELO Dokumententyp.
MergeImagesEx	Kopiert eine Tiff Datei in eine andere Tiff Datei. Diese Funktion kann zur dauerhaften Schwärzung oder zum dauerhaften Aufbringen von Stempeln verwendet werden.

SeparateTiffFile	Mit der Funktion SeparateTiffFile können Sie eine Multipage Tiff-Datei in Einzeldateien aufsplitten. Diese können dabei automatisch, über Trennseiten gesteuert, wieder zu Teildokumenten zusammengefasst werden. Für die Einzeldateien wird als Dateiname der Originalname verwendet. Dieser wird jeweils um ein Unterstrich und der Zeilennummer ergänzt (beginnend mit 0). Aus der Datei XYZ.TIF werden also die Einzeldateien XYZ_0.TIF, XYZ_1.TIF ... , diese werden im gleichen Verzeichnis abgelegt.
SetCookie	Die Funktion SetCookie setzt den Wert eines Cookie-Eintrags. Falls das Cookie noch nicht existiert wird es automatisch erzeugt. Die Funktionen Get/SetCookie sind primär dazu gedacht, daß ELO Scripting Macros dauerhaft Informationen in ELO hinterlegen können. Der Cookie-Speicher wird erst beim Beenden von ELO gelöscht.
SplitFileName	Die Funktion SplitFileName ermittelt aus einem Dateipfad (Laufwerk – Pfad – Dateiname bzw. Server – Pfad –Dateiname) den Pfadanteil mit Laufwerk oder Server, den Dateinamen oder die Dateikennung.

CheckIn/ Out

CheckInEx	Über diese Funktion kann eine Datei mit Versionsnummer und Kommentar wieder eingecheckt werden.

CheckInOutFileName	Innerhalb des Ereignisses „Beim Aus-/ Einchecken eines Dokuments“ enthält dieses Property den Dateinamen des aktuellen Eintrags.
CheckInOutObjID	Innerhalb des Ereignisses „Beim Aus-/ Einchecken eines Dokuments enthält dieses Property die interne ELO Objektnummer.
CheckOut	Sperrt ein ELO Dokument und fügt eine Kopie der Dokumentendatei in das CheckOut Verzeichnis ein. Optional kann auch die zuständige Applikation zur Bearbeitung gestartet werden.
DoCheckInOut3	Ruft den Dialog zum Ein- und Auschecken von Dokumenten auf.

Formularerkennung/ OCR

GetOcrRectList	Diese Funktion gibt eine Liste von Trefferrechtecken zu einem gesuchten Wort zurück. Sie kann z.B. für eine Treffermarkierung nach der Suche in einem Dokument verwendet werden.
OcrAddRect	Fügt ein weiteres Rechteck in die OCR Erkennungs-Rechteckliste ein. Zum Aufbau einer Rechteckliste sollte immer zuerst mit OcrClearRect ein definierte Zustand hergestellt werden, danach kommt eine Folge von maximal 32 OcrAddRect Befehlen. Das Rechteck wird in Form eines Strings mit den linken, oberen Koordinaten und den rechten, unteren Koordinaten angegeben. Alle Werte werden in Promille ausgedrückt (z.B.: 0,0,999,999 ist die ganze Seite).

OcrAnalyzeEx	Die Funktion OcrAnalyze übergibt den Namen der auszuwertenden Datei und liest anhand der voreingestellten Rechteckliste die Texte in die interne Textliste ein. Diese Liste kann über den Befehl OcrGetText ausgelesen werden.
OcrClearRect	Löscht die interne Rechteckliste.
OcrGetPattern	Liefert einen erkannten Teiltext eines Musters zurück. Da das interne Pattern Matching weniger leistungsfähig ist als die regulären Ausdrücke, ist diese Funktion nur noch aus Kompatibilitätsgründen vorhanden und sollte für Neuentwicklungen nicht mehr eingesetzt werden.
OrcGetText	Nach der OCR Analyse stehen in einem Textfeld die erkannten Teile der Recheckliste zur Verfügung. Über den Befehl OcrGetText können Sie diese Texte auslesen, OcrGetText(0) liefert den Text zum ersten Rechteck, OcrGetText(1) zum zweiten usw…
OcrPattern	Siehe OcrGetPattern.

AutoDialog anzeigen

Ein AutoDialog muss zuerst durch CreateAutoDlg erstellt werden. Anschließend wird er mit Elementen gefüllt (Textfelder, Eingabefelder, CheckBox, RadioButton). Mittels ShowAutoDlg kann er dann angezeigt werden. Nach der Anzeige können die Anwendereingaben durch GetAutoDlgValue ausgelesen werden.

AddAutoDlgControl	Erstellt ein neues Element im aktuellen AutoDialog.

AutoDlgResult	Gibt alle Eingabewerte des AutoDialogs zurück. Die einzelnen Eingaben sind jeweils durch einen Zeilenwechsel getrennt.
CreateAutoDlg	Erzeugt einen neuen, leeren Dialog.
GetAutoDlgValue	Gibt einen Eingabewert zurück.
ShowAutoDlg	Zeigt den Autodialog an und wartet, bis er vom Anwender gefüllt und geschlossen wird.

Anwenderverwaltung

FindUserEx	Ermittelt zu einem Anwendernamen die interne ELO Anwendernummer.
LoadUserName	Ermittelt zu einer Anwendernummer den Anwendernamen.
Login	Anmeldung am System unter dem angegebenen Anwendernamen. Dieser Befehl wird auch zur Abmeldung verwendet, als Anwendername wird dann „LOGOUT“ verwendet.
LookupUserName	Ermittelt zu einem Anwendernamen die interne ELO Anwendernummer.
ReadUser	Die Funktion ReadUser liest einen Anwenderdatensatz aus der Systemdatei ein. Administratoren können beliebige Anwender lesen und bearbeiten, Subadministratoren können nur eigene Anwender lesen und bearbeiten.

SelectUserEx	Über die Funktion SelectUserEx können Sie einen Dialog zur Auswahl eines Anwenders aufrufen. Es wird eine Liste aller Anwender angezeigt (optional ohne eigenen Eintrag) und Sie erhalten als Rückgabe die ausgewählte Anwendernummer oder eine –1 bei Abbruch.
UserFlags	Das Property UserFlags enthält die Rechte des Anwenders (als Bitmaske). Ein Administrator kann dabei beliebige Rechte zuteilen, ein Subadministrator maximal seine eigenen Rechte. Falls er mehr zuweist, als er selber besitzt, führt das nicht zu einer Fehlermeldung, die Einstellung wird automatisch angepaßt.
UserId	Über das Property UserId können Sie die Nummer des aktuell aktiven Anwederdatensatzes ermitteln.
UserKeys	Das Property UserKeys enthält die Schlüssel des Anwenders (als Zahlenfolge mit Komma getrennt, paarweise die Schlüsselnummer und das Recht). Ein Administrator kann dabei beliebige Schlüssel zuteilen, ein Subadministrator maximal seine eigenen Schlüssel. Es liegt in der Verantwortung des Skript-Programmierers dafür zu sorgen, dass hier wirklich nur verfügbare Schlüssel zugewiesen werden.
UserName	Über das Property UserName können Sie den Anwendername des aktuell geladenen Anwenderdatensatzes lesen oder ändern.
UserParent	Über das Property UserParent, kann einem Anwender ein (Sub)Administrator zugewiesen werden. Dieser (Sub)Administrator hat dann das Recht, die UserDaten des Anwenders zu ändern.

WriteUser	Die Funktion WriteUser schreibt den aufbereiteten Anwenderdatensatz in die Datenbank. Administratoren können beliebige Anwender schreiben, Subadministratoren können nur eigene Anwender oder neue Anwender schreiben.

Verschlagwortungsmasken

LookupMaskName	Ermittelt zu dem Namen einer Verschlagwortungsmaske die interne ELO Maskennummer.
ReadObjMask	Mit dieser Funktion können Sie eine Maske (Dokumententyp) in das interne ELO Objekt einlesen.
WriteObjMask	Mit dieser Funktion können Sie eine Maskendefinition schreiben. Die Maske muss vorher mit ReadObjMask gelesen worden sein. Wenn Sie eine neue Maske anlegen wollen, so muss diese mit ReadObjMask(9999) initialisiert werden.

Farbverwaltung

Farben werden im ELOoffice nicht direkt mit einem RGB Farbwert angegeben. Es gibt eine Tabelle von Farbmarkern, jeder hat einen Namen und einen Farbwert. Intern in der Verschlagwortung wird mit dem Index in diese Farbtabelle gearbeitet. Der Index darf im Bereich von 1..60 liegen.

In der OLE Schnittstelle gibt es ein Farbobjekt. Es wird mittels ReadColorNo gelesen oder initialisiert. Über die Properties ColorInfo und ColorName können die Eigenschaften RGB-Farbwert und Name gesetzt werden.

Anschließend wird diese Information mittels WriteColorInfo in die Datenbank geschrieben.

ColorInfo	RGB Farbwert des Eintrags
ColorName	Name des Eintrags
ReadColorInfo	Mit dieser Funktion können Sie eine Farbdefinition in das aktuelle Elo-Farbobjekt einlesen. Auf die Farbeinstellungen können Sie dann mittels der Properties ColorInfo und ColorName zugreifen.
WriteColorInfo	Mit dieser Funktion können Sie eine Farbdefinition aus dem aktuellen ELO-Farbobjekt in die Datenbank schreiben.

Wiedervorlagetermine bearbeiten

Wiedervorlagetermine werden über die Funktionen ReadWv und WriteWv gelesen und geschrieben. Bestehende Termine werden über die Wiedervorlage-Id gelesen, neue Termine mittels der Pseudo-Id 0 erzeugt. Auf den Termin kann dann mit folgenden Properties lesend und schreibend zugegriffen werden:

CollectWvEx	Sammelt die Liste der Aktivitäten und Wiedervorlagetermine eines Anwenders. Die Ergebnisse können mittels GetListEntry(i) abgefragt werden.
DeleteWv	Löscht den angegebenen Wiedervorlagetermin.
DeleteWvLine	Entfernt einen Eintrag aus der Liste der angezeigten Wiedervorlagetermine. Der Termin selber wird aber nicht gelöscht. Diese Funktion kann zum Filtern der Aufgabenansicht verwendet werden.

EditWv	Aktiviert den Dialog zur Bearbeitung eines Wiedervorlagetermins.
FilterText	Hierüber kann der Filter für die Aufgabenansicht gelesen oder verändert werden.
FindFirstWv	Ermittelt den ersten Wiedervorlagetermin zu einem ELO Objekt.
FindNextWv	Ermittelt den nächsten Wiedervorlagetermin zu einem ELO Objekt.
ReadWv	Liest einen Wiedervorlagetermin (bestimmt durch den Parameter WvIdent) in den internen Wv-Speicher ein. Dieser Termin kann dann mit den verschiedenen Property-Funktionen ausgelesen werden. Ein WvIdent 0 bewirkt, daß der interne Wv-Speicher gelöscht wird, dieser Aufruf ist vor dem Anlegen eines neuen Termins notwendig.
ReloadWv	Aktualisiert die Aufgabenansicht.
WriteWv	Schreibt einen Wiedervorlagetermin (bestimmt durch den Parameter WvIdent) aus dem internen Wiedervorlagenobjekt in die Datenbank. Ein WvIdent Wert 0 bewirkt, dass ein neuer Wv-Termin angelegt wird. Vor dem Füllen eines neuen Termins muss das interne Wiedervorlagenobjekt durch einen ReadWv Aufruf mit Parameter 0 gelöscht werden.
WvIdent	(ReadOnly) Interne ELO Wiedervorlagennummer des aktuellen Termins.
WvParent	ELO Objektnummer des Ordners oder Dokuments zu dem dieser Termin erzeugt wurde.

WvUserFrom	ELO Anwendernummer der Person, die den Termin erzeugt hat.
WvUserOwner	Anwendernummer der Person, die diesen Termin bearbeiten soll.
WvCreateDate	Datum an dem der Termin erzeugt wurde. Dieses Property muss nicht gefüllt werden, es wird dann beim Speichern automatisch auf das aktuelle Tagesdatum gesetzt.
WvDate	Fälligkeitsdatum des Termins.
WvListInvalidate	Dieser Aufruf aktualisiert die Aufgabenliste.
WvPrio	Priorität des Termins. Es stehen die Werte 1 (Wichtig), 2 (Normal) und 3 (weniger Wichtig) zur Verfügung. Alle anderen Werte werden automatisch auf 2 (Normale Priorität) gesetzt.
WvParentType	Typ des zugeordneten Ordners (1..32) oder Dokuments (254..285). Wenn ein Termin per Skript erzeugt wird, sollten Sie darauf achten, dass Sie diesen Typ aus dem Originaleintrag per GetObjTypeEx übernehmen. Wenn es hier Unterschiede zwischen dem Eintrag und dem Termin gibt, kann das beim Anwender zu erheblichen Irritationen führen.
WvShort	Kurzbezeichnung des Termins. Bei manuell angelegten Terminen wird hier als Vorgabe die Kurzbezeichnung des Ordners oder Dokuments eingetragen. Der Anwender kann diesen Text aber vor dem Speichern beliebig abändern.
WvDesc	Arbeitsanweisung zu dem Termin.

Anhang A: Wichtige Konstanten

Allgemeine Systemkonstanten

vbCr	Zeilenumbruch durch ein „Carriage Return“. Diese Art des Zeilenumbruchs wird vorwiegend im Unix Bereich verwendet.
vbCrLf	Zeilenumbruch durch ein „Carriage Return und Line Feed“. Im Windows Umfeld ist das der normale Zeilenumbruch.
vbFormFeed	Seitenumbruch Zeichen
vbTab	Tabulatorzeichen

Konstanten für die Funktion MsgBox

Auswahl der Schaltflächen für die Message Box über den Parameter Flags. Einer dieser Parameter muss angegeben werden, er entscheidet über die prinzipielle Art der Anzeige.

vbOkOnly	Message Box mit Ok Schaltfläche
vbOkCancel	Message Box mit Ok und Abbruch Schaltflächen
vbAbortRetryIgnore	Message Box mit Abbruch, Retry und Ignore Schaltflächen
vbYesNo	Message Box mit Ja und Nein Schaltflächen
vbYesNoCancel	Message Box mit Ja, Nein und Abbruch Schaltflächen

vbRetryCancel	Message Box mit Wiederholen und Abbruch Schaltflächen

Zusätzlich kann zur Verdeutlichung der Aktion noch ein Symbol in die Message Box aufgenommen werden.

vbCritical	Stoppsymbol anzeigen
vbQuestion	Fragezeichen anzeigen
vbExclamation	Warnung anzeigen
vbInformation	Informationssymbol anzeigen

Über einen weiteren Zusatz kann man bestimmen, welche Schaltfläche die Standardschaltfläche ist. Diese wird durch einen Klick auf die Return Taste ausgelöst.

vbDefaultButton1	Die erste Schaltfläche als Standard verwenden
vbDefaultButton2	Die zweite Schaltfläche als Standard verwenden
vbDefaultButton3	Die dritte Schaltfläche als Standard verwenden

Rückgabewerte der MsgBox

vbOk	Ok Schaltfläche wurde betätigt
vbCancel	Abbruch Schaltfläche wurde betätigt
vbAbort	Abbruch Schaltfläche wurde betätigt
vbRetry	Wiederholen Schaltfläche wurde betätigt

vbIgnore	Ignorieren Schaltfläche wurde betätigt
vbYes	Ja Schaltfläche wurde betätigt
vbNo	Nein Schaltfläche wurde betätigt

Anhang B: Automatische Skripte

In der folgenden Tabelle sind die wichtigsten automatischen Skriptereignisse aufgeführt und kurz erläutert. Eine vollständige Auflistung finden Sie in der OLE Automation Schnittstellenbeschreibung.

ELO_BUZZLIST	Vor der Anzeige einer Stichwortliste wird dieses Skriptereignis aufgerufen. Ein Skript kann hierüber Einfluss auf die angezeigten Stichwörter nehmen.
ELO_FREEZE	Beim Konvertieren von Dokumenten in das TIFF oder PDF Format ruft ELOoffice die Windows Systemfunktion ShellExecute(Print) auf. Falls Dokumententypen vorhanden sind, die über diese Funktion nicht korrekt ausgedruckt werden, kann in diesem Skript eine andere Druckfunktion aktivieren. Wenn das Skript eine eigene Druckausgabe durchführt, muss der ActionKey Wert anschließend auf 1 gesetzt werden damit ELO nicht zusätzlich noch einen Druck durchführt.
ELO_OUTLOOK	Beim Anlegen eines Wiedervorlagetermins wird dieses Skriptereignis aufgerufen, wenn es in den Optionen / Aufgaben konfiguriert wurde. Dieses Skript ist dann verantwortlich dafür, dass im Outlook eine Aufgabe angelegt oder eine Mail verschickt wird.
ELO_START	Dieses Skriptereignis wird beim Betreten oder Verlassen eines Archivs aufgerufen (ActionKey = 1 beim Betreten, ActionKey = 2 beim Verlassen). Es eignet sich für Initialisierungen oder Aufräumarbeiten.

Anhang C: ActionKeys

Das ActionKey Propery kann in Ereignisbehandlungsroutinen aufgerufen werden. Dort wird der aktuelle Systemzustand signalisiert, wenn das gleiche Event mehrfach innerhalb einer Aktion aufgerufen werden kann.

Beispiel: beim Betreten der Verschlagwortung ist der Wert des ActionKey 20 (Event: Beim Bearbeiten der Verschlagwortung, aktueller Aufruf: Maske betreten), beim Verlassen 21.

Dialog: Dokument bearbeiten	Dialog in Postboxansicht aufgerufen, Dokumententyp noch nicht definiert	19
	Maske betreten	20
	Maske mit Speichern verlassen	21
	Maske mit Abbruch verlassen	22
	Dokumententyp geändert	23
	Schaltflächenliste abfragen	24
Nach bearbeiten Maskenfeld	Feld Kurzbezeichnung betreten	10
	Feld Kurzbezeichnung verlassen	11
	Feld Memo betreten	12
	Feld Memo verlassen	13

	Feld Datum betreten	14
	Feld Datum verlassen	15
	Index-Eingabezeile n betreten	1000+n (n=0..49)
	Index-Eingabezeile n verlassen	2000+n
	Anwenderdefinierten Button betätigt	3000+n
Vor dem Importieren/ Exportieren	Vor dem Importieren eines Aktenstrukturelements	1
	Vor dem Exportieren eines Aktenstrukturelements	2
Beim Aus-/Einchecken	Nach dem Auschecken	30
	Vor dem Einchecken, nach der Versionsabfrage	31
	Nach dem Einchecken	32
	Vor dem Auschecken	33
	Vor dem Einchecken, vor der Versionsabfrage	34
	Vor dem Verwerfen	38
	Vor dem Aktivieren/Anzeigen	80
	Vor dem Ausdrucken	81
	Register CheckIn/Out	1001 … 1005

Anwender lesen/speichern	Unmittelbar vor dem Abspeichern	20000
	Nach dem Abspeichern	20001
	Nach dem Einlesen	20010
Stichwortliste	Vor dem Bearbeiten der Stichwortliste	1
	Vor der Anzeige der Stichwortliste	2
Wiedervorlage	Termin als Aufgabe für anderen Anwender	1
	Termin als Aufgabe für sich selber	2
	Termin als e-mail	3
ClickOn Event	Auswahl eines Buttons oder Menüeintrags	40
Dokument einfrieren	Vor dem Ausdrucken	0
Suchen („Vor dem Sammeln der Rechercheliste“)	Vor der Suche	1
	F7-Gruppensuche	2
	Kombinierte Gruppensuche	3
	Nach der Suche	4

Anhang D: Klassen und Funktionen zur Wiederverwendung

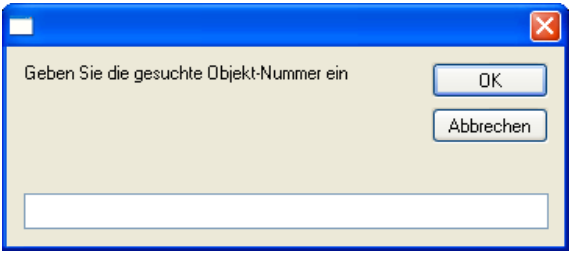
Im Folgenden werden einige Klassen und Funktionen aus dem Buch aufgeführt, die Sie leicht in eigene Skripte übernehmen können. Da es keinen Include Mechanismus für Subroutinen gibt, müssen diese per Copy & Paste kopiert werden. Das ist aus der Sicht des Software Engineering zwar nicht optimal, es geht aber nicht anders.

InputBox

Die normale Windows InputBox ist ein sehr einfacher und grauer Systemdialog. Diese kann man durch einen Dialog aus ELO ersetzen.

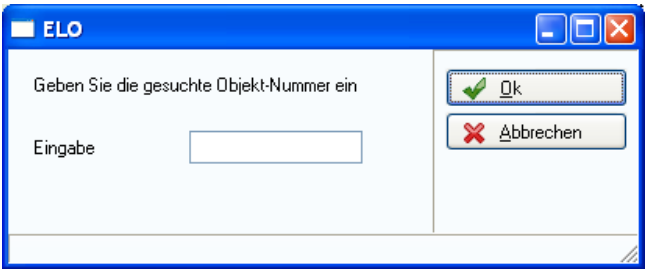
Statt

```
set ELO = CreateObject("ELO.office")
UserInput = InputBox("Geben Sie die gesuchte Objekt-Nummer ein")
```



kann man folgenden Aufruf verwenden.

```
set ELO = CreateObject("ELO.office")
UserInput = EloInputBox("Geben Sie die gesuchte Objekt-Nummer ein",
"ELO")
```



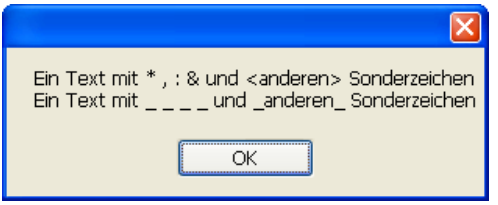
Dafür muss man nur die folgende Funktion in das Skript kopieren

```
function EloInputBox(Message, Titel)
  ELO.CreateAutoDlg(Titel)
  call ELO.AddAutoDlgControl(1, 0, Message, "")
  call ELO.AddAutoDlgControl(4, 2, "Eingabe", " ")
  if ELO.ShowAutoDlg = 1 then
    EloInputBox = ELO.GetAutoDlgValue(2)
  else
    EloInputBox = ""
  end if
end function
```

Dateiname aus der Kurzbezeichnung bilden

Bei vielen Aufgaben wird ein Dateiname benötigt, der oft aus der Kurzbezeichnung gewonnen werden soll. Da die ELO Kurzbezeichnung aber eine Reihe von Zeichen erlaubt, die in Dateinamen nicht vorhanden sind, ist eine direkte Verwendung ausgeschlossen. Die folgende kleine Hilfsfunktion kann zu Konvertierung der Kurzbezeichnung in einen Dateinamen verwendet werden. Sie kann direkt mit dem Property ObjShort oder einer Textvariablen aufgerufen werden und liefert einen bereinigten Dateinamen zurück.

```
Kurzbezeichnung = "Ein Text mit * , : & und <anderen> Sonderzeichen"
FileName = PrepareFileName(Kurzbezeichnung)
MsgBox Kurzbezeichnung & vbCrLf & FileName
```



Zur Verwendung müssen Sie nur folgenden Codeabschnitt in Ihr Skript kopieren:

```
Function PrepareFileName(Text)
  Dim Reg
  Set Reg = New RegExp

  Reg.Global = True
  Reg.Pattern = "[\*\:\,\\\&\t\|\r\n<>]"
  PrepareFileName = Reg.Replace(Text, "_")
End Function
```

Lesen/ Schreiben nach Gruppenname

Diese beiden Hilfsroutinen lesen oder schreiben ein Indexzeile. Diese wird aber nicht über die Zeilennummer sondern über den Gruppennamen bestimmt. Falls der Gruppenname in der aktuellen Verschlagwortung nicht existiert, wird ein Leerstring gelesen und die Schreiboperation ignoriert.

```
' Schreibt den Wert "AttribValue" in die
' Indexzeile mit dem Gruppennamen "AttribName".
' Der erste Parameter enthält das ELO Objekt.
Sub SetObjAttribByName(ELO, AttribName, AttribValue)
  Dim i, LocalName

  LocalName = LCase(AttribName)
  For i = 0 to 49
    If LCase(ELO.GetObjAttribKey(i)) = LocalName Then
      Call ELO.SetObjAttrib(i, AttribValue)
      Exit For
    End If
  Next
```

```
End Sub

' Liest den aktuellen Wert der Indexzeile mit
' dem Namen "AttribName". Der erste Parameter
' enthält das ELO Objekt
Function GetObjAttribByName(ELO, AttribName)
  Dim i, LocalName

  GetObjAttribByName = ""
  LocalName = LCase(AttribName)
  For i = 0 to 49
    If LCase(ELO.GetObjAttribKey(i)) = LocalName Then
      GetObjAttribByName = ELO.GetObjAttrib(i)
      Exit For
    End If
  Next
End Function
```

ScriptWalk Klasse für Teilarchive

Diese Klasse führt einen Durchlauf durch ein Teilarchiv, beginnend ab einem definierten Startpunkt, aus. Für die Aktionen, die auf jedem Eintrag ausgeführt werden sollen, muss eine Subroutine ScriptWalk_Process angelegt werden. Diese wird als Funktionszeiger an die Methode Walk übergeben und für jeden durchlaufenen Eintrag aufgerufen.

```
Sub ScriptWalk_Process
  …
End Sub
```

Der Aufruf findet über die Methode Walk statt, als Parameter werden der Startpunkt, die maximale Durchlauftiefe und die Callback Routine mitgegeben.

```
Set Exporter = new ScriptWalk
Call Exporter.Walk(StartId, MaxLevel, Callback)
```

Folgende Methoden werden von der Klasse zur Verfügung gestellt

Walk(StartId, MaxLevel, Callback)	Startet einen Durchlauf über den Teilbaum, beginnend bei StartId. Die Suchtiefe kann über den Parameter MaxLevel begrenzt werden, damit sich das Skript bei zyklischen Archivstrukturen nicht aufhängt. Die Callback Routine wird dann für jeden durchlaufenen Eintrag aufgerufen.
GetActPath	Gibt die relative Position innerhalb des Teilbaums zurück, beginnend ab der Startposition. Im Standard wird der Pfad aus der Kurzbezeichnung zusammengesetzt, diese kann aber in der ScriptWalk_Process Callback Routine über das Property LevelName geändert werden.
LevelName	(Property, Lesen/ Schreiben) Gibt den Namen des aktuellen Eintrags an. Er wird durch die Kurzbezeichnung initialisiert, kann aber in der ScriptWalk_Process Callback Routine geändert werden. Diese Änderungen fließen dann auch in den gebildeten Pfad ein, der über GetArcPath gelesen werden kann.

```
Class ScriptWalk
  Private Level
  Private LocalMaxLevel
  Private ActPath
  Private ProcessFunction
  Private M_LevelName

  Public Sub Walk(StartId, MaxLevel, Callback)
    Level = 0
    ActPath = ""
    LocalMaxLevel = MaxLevel
    Set ProcessFunction = Callback
    Call LocalWalk(StartId)
  End Sub

  Public Function GetActPath
```

```
  GetActPath = ActPath
 End Function

 Public Property Get LevelName
  LevelName = M_LevelName
 End Property

 Public Property Let LevelName(NewName)
  M_LevelName = NewName
 End Property

 Private Sub LocalWalk(StartId)
  Dim ChildListString, ChildList, i, Msg, OldPath

  ' Zuerst den Eintrag selber einlesen und prüfen
  If ELO.PrepareObjectEx(StartId, 0, 0) < 0 Then
   ' Der Eintrag konnte nicht gelesen werden, hier abbrechen
   Exit Sub
  End If

  M_LevelName = ELO.ObjShort
  Call ProcessFunction

  ' Nun die Nachfolgerliste prüfen. Nur Ordner haben Nachfolger
  If ((ELO.ObjTypeEx <= MaxFolder) Or (ELO.ObjTypeEx = 9999)) _
      and (Level < LocalMaxLevel) Then
    ChildListString = ELO.CollectChildList(StartId)
    If ChildListString <> "" Then
      ChildList = Split(ChildListString, ":")
      Level = Level + 1
      OldPath = ActPath
      ActPath = ActPath & "\" & M_LevelName
      For i = 0 to UBound(ChildList)
       If ChildList(i) <> "" Then
         Call LocalWalk(ChildList(i))
       End If
      Next
      Level = Level - 1
      ActPath = OldPath
    End If
  End If
 End Sub
End Class
```

Ein Fortschrittsbalken im Internet Explorer

Der Fortschrittsbalken wird so verwendet, dass nach der Erzeugung des Objekts zuerst die Methode Show zur Anzeige aufgerufen wird. Wenn die Bearbeitung in einer genau bekannten Anzahl von Schritten abläuft, dann die Weiterschaltung einfach mit „Increment(MessageText)“ erfolgen, andernfalls muss per Update(AktuellerStand, MessageText) die jeweilige aktuelle Position mitgeteilt werden. Am Ende wird die Anzeige mittels Hide beendet.

Es werden folgende Funktionen zur Verfügung gestellt

Show(Title, Anzahl, Message)	Blendet den Internetexplorer ein, zeigt die Startmeldung und den Titel an und startet die Animation. Die Anzahl gibt den Wert für die 100% Anzeige an. Der Parameter Message muss gültigen HTML Code enthalten.
Update(Stand, Message)	Aktualisiert den Meldungstext um den Anwender über den Fortschritt der Arbeiten zu Informieren. Über den Parameter Stand wird die aktuelle Position angegeben. Wenn die Anzahl aus Show() mit 500 initialisiert wurde, wird ein Stand von 100 eine 20% Anzeige auslösen. Der Parameter Message muss gültigen HTML Code enthalten.
Increment(Message)	Schaltet die Anzeige um einen Schritt weiter (nicht um ein Prozent). Der Parameter Message muss gültigen HTML Code enthalten.
Hide(Message, Delay)	Zeigt für eine einstellbare Zeit die Abschlussnachricht an und entfernt die Anzeige dann wieder. Der Parameter Message muss gültigen HTML Code enthalten.

```
Class ExplorerStatusBar

Private Explorer, Total, Count

' Zeigt die Statusanzeige an und speichert den 100% Wert
Public Sub Show(Titel, MaxValue, StartMessage)
  Set Explorer = CreateObject("InternetExplorer.Application")
  Explorer.Navigate "about:blank"
  Explorer.ToolBar = 0
  Explorer.StatusBar = 0
  Explorer.Width=370
  Explorer.Height = 160
  Explorer.Left = 0
  Explorer.Top = 0

  Do While (Explorer.Busy)
    call ELO.Sleep(0, 200)
  Loop

  Explorer.Visible = 1
  Explorer.Document.Body.InnerHTML = StartMessage
  Explorer.Document.Title = Titel

  Total = MaxValue
  Count = 0
End Sub


' Verlängert die Statusanzeige um einen Schritt
Public Sub Increment(Message)
    call Update(Count + 1, Message)
End Sub


' Aktualisiert die Länge der Statusanzeige auf eine neuen Wert
Public Sub Update(NewStatusCount, Message)
    Dim Prozent
    Count = NewStatusCount
    if Count > Total then Count = Total
    Prozent = CLng((Count / Total) * 100)
    Explorer.Document.Body.InnerHTML = _
      "<div style=""border:solid 1px;width:302px""><div style=""width:" _
```

```
    & 3 * prozent & "px;background-color:#ffeedd""></div></div><div>" _
    & Prozent & "% (" & Count & " of " & Total _
    & ")</div><div> </div><div>" & Message & "</div>"
End Sub


' Schließt die Statusanzeige wieder
Public Sub Hide(ReadyMessage, ShowDelay)
  Explorer.Document.Body.InnerHTML = ReadyMessage
  call ELO.Sleep(0, ShowDelay)
  Explorer.Quit
End Sub

End Class
```

Eine Warteanzeige im Internet Explorer

Die Warteanzeige wird so verwendet, dass nach der Erzeugung des Objekts zuerst die Methode Show zur Anzeige aufgerufen wird. Während der Bearbeitung wird regelmäßig Update mit der Information über den aktuellen Bearbeitungsstand aktiviert. Am Ende wird die Anzeige mittels Hide beendet.

```
Set Status = new ExplorerWaitBar
call Status.Show(…)

For Each …
  call Status.Update(ObjService.Name)
Next

call Status.Hide("Liste vollständig abgearbeitet.", 1000)
```

Es werden folgende Funktionen zur Verfügung gestellt

Show(Title, Message)	Blendet den Internetexplorer ein, zeigt die Startmeldung und den Titel an und startet die Animation. Der Parameter Message muss gültigen HTML Code enthalten.

Update(Message)	Aktualisiert den Meldungstext um den Anwender über den Fortschritt der Arbeiten zu Informieren. Der Parameter Message muss gültigen HTML Code enthalten.
Hide(Message, Delay)	Zeigt für eine einstellbare Zeit die Abschlussnachricht an und entfernt die Anzeige dann wieder. Der Parameter Message muss gültigen HTML Code enthalten.

```
' Hier beginnt der Funktionsbereich für die Klasse der Statusanzeige
' Die Statusanzeige wird durch Methodenaufrufe in folgender
Reihenfolge
' verwendet
'
'   Show
'   mehrfaches Update
'   Hide

Class ExplorerWaitBar

Private Explorer

' Zeigt die Statusanzeige an
Public Sub Show(Titel, StartMessage)
  Set Explorer = CreateObject("InternetExplorer.Application")
  Explorer.Navigate "about:blank"
  Explorer.ToolBar = 0
  Explorer.StatusBar = 0
  Explorer.Width=370
  Explorer.Height = 160
  Explorer.Left = 0
  Explorer.Top = 0

  Do While (Explorer.Busy)
    call ELO.Sleep(0, 200)
  Loop
```

```
  Explorer.Visible = 1
  call PrepareHtml(Titel, StartMessage)
End Sub



' Aktualisiert die Warteanzeige
Public Sub Update(Message)
  Dim Doc, Div

  Set Doc = Explorer.Document
  Set Div = Doc.getElementById("msg")
  Div.innerHTML = Message
End Sub


' Schließt die Statusanzeige wieder
Public Sub Hide(ReadyMessage, ShowDelay)
  Explorer.Document.Body.InnerHTML = ReadyMessage
  call ELO.Sleep(0, ShowDelay)
  Explorer.Quit
End Sub

Private Sub PrepareHtml(Title, StartMessage)
  Dim Hdr, Body, Start, Doc

  Hdr ="<head><title>" & Title & "</title>" & _
      "<script type=""text/javascript"">" & _
      "var leftpos = 0;" & _
      "var innerpos = 0;" & _
      "var direction = 1;" & _
      "function tick() {" & _
      " var outer = document.getElementById(""outer"");" & _
      " var inner = document.getElementById(""inner"");" & _
      " if (inner && outer) {" & _
      " if (direction == 1) {" & _
      "   if (innerpos < 72) {" & _
      "     innerpos++;" & _
      "     inner.style.left = innerpos + ""px"";" & _
      "   } else {" & _
      "     leftpos++;" & _
      "     outer.style.left = leftpos + ""px"";" & _
      "     if (leftpos > 120) {" & _
      "       direction = -1;" & _
```

```
"    }" & _
"   }" & _
" } else {" & _
"   if (innerpos > 0) {" & _
"    innerpos--;" & _
"    inner.style.left = innerpos + ""px"";" & _
"   } else {" & _
"    leftpos--;" & _
"    outer.style.left = leftpos + ""px"";" & _
"    if (leftpos < 1) {" & _
"     direction = 1;" & _
"    }" & _
"   }" & _
" }" & _
"}" & _
"}" & _
"</script></head>"

Body = "<body scroll=""no"" onload=""setInterval('tick()', 20);"">" & _
    "<div style=""border:solid 1px;width:203px;color:#808080"">" & _
    "<div id=""outer"" style=""position:relative;left:0px;" & _
    "width:80px;background-color:#f0f0fa;height:30px"">" & _
    "<div id=""inner"" style=""position:relative;left:0px;width:8px;" & _
    "background-color:#00ff80;height:30px"">" & _
    "<div style=""position:relative;left:0px;width:4px;" & _
    "background-color:#ffc840;height:30px"">" & _
    " </div></div></div></div><div id=""msg"">" & _
    StartMessage & "</div></body>"

Start = "<html>" & Hdr & Body & "</html>"
Set Doc = Explorer.Document
Doc.Open
Call Doc.Write(Start)
Doc.Close
End Sub

End Class
```

Quicksort für einfache Datentypen und Objekte

Die Quicksort Funktion sortiert ein Array oder Teilarray aus Strings oder Objekten. In der Version SortAll werden als Parameter das zu sortierende Array und die Sortierfunktion übergeben. Die Version Sort kann zusätzlich

noch die Nummer des ersten und des letzten zu sortierenden Felds erhalten, so dass auch Teilarrays sortiert werden können.

Der Aufruf sieht dann so aus:

```
Call Sort(Data, 0, UBound(Data), GetRef("CompareIsLowerNoCase"))
```

Oder so:

```
Call SortAll(Data, GetRef("CompareIsLowerNoCase"))
```

Die angegebenen Sortierfunktionen CompareIsLowerNoCase und CompareIsLowerWithCase können für eine Textsortierung von Strings mit oder ohne Berücksichtigung der Groß/ Kleinschreibweise verwendet werden. Für die Verwendung eigener Objekte muss eine eigene Vergleichsfunktion erstellt werden, die True liefert wenn der erste Parameter kleiner ist als der zweite und False im anderen Fall.

```
' Die Vergleichsfunktion "NoCase" führt
' einen case insensitiven Vergleich aus.
' Das ergibt für Texte eine natürliche
' Sortierung
Function CompareIsLowerNoCase(Val1, Val2)
  CompareIsLowerNoCase = StrComp(Val1, Val2, vbTextCompare) < 0
End Function

' Die Vergleichsfunktion "WithCase" führt
' einen binären Vergleich aus. Das hat zur
' folge, dass Kleinbuchstaben komplett hinter
' den Großbuchstaben liegen A..Za..z
Function CompareIsLowerWithCase(Val1, Val2)
  CompareIsLowerWithCase = Val1 < Val2
End Function


' Die Sortierfunktion SortAll erhält als Parameter
' das zu sortierende Array und die Vergleichsfunktion
' Es wird immer das komplette Array sortiert
```

```
Function SortAll(ByRef Data, ByRef CompareIsLower)
  Call Sort(Data, 0, UBound(Data), CompareIsLower)
End Sub

' Die Sortierfunktion Sort erhält als Parameter
' das zu sortierende Array, erstes und letztes
' zu sortierende Element und die Vergleichsfunktion
Function Sort(ByRef Data, ByRef Low, ByRef High, ByRef
CompareIsLower)
  Dim MidVal, Swap, CurLow, CurHigh, Midpoint, IsObject

  If High <= Low Then
    Exit Function
  End If

  CurLow = Low
  CurHigh = High
  Midpoint = (Low + High) \ 2
  If VarType(Data(Midpoint)) = vbObject Then
    IsObject = True
    Set MidVal = Data(Midpoint)
  Else
    IsObject = False
    MidVal = Data(Midpoint)
  End If

  Do While (CurLow <= CurHigh)
    Do While CompareIsLower(Data(CurLow), MidVal)
      CurLow = CurLow + 1
      If CurLow = High Then
        Exit Do
      End If
    Loop

    Do While CompareIsLower(MidVal, Data(CurHigh))
      CurHigh = CurHigh - 1
      If CurHigh = Low Then
        Exit Do
      End If
    Loop

    If (CurLow <= CurHigh) Then
      If IsObject Then
        Set Swap = Data(CurLow)
        Set Data(CurLow) = Data(CurHigh)
```

```
      Set Data(CurHigh) = Swap
    Else
      Swap = Data(CurLow)
      Data(CurLow) = Data(CurHigh)
      Data(CurHigh) = Swap
    End If

    CurLow = CurLow + 1
    CurHigh = CurHigh - 1
  End If

 Loop

 If Low < CurHigh Then
  Call Sort(Data, Low, CurHigh, CompareIsLower)
 End If

 If CurLow < High Then
  Call Sort(Data, CurLow,High, CompareIsLower)
 End If
End Function
```

Stichwortregister

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z