Erschienen im
FileMaker Magazin
www.filemaker-magazin.de

Robert Hermann
(Jg. 1961) studierte Drucktechnik (FH), gelernter Schriftsetzer, 1992 Wechsel in die Technik mit Fokus Apple-Computer. FileMaker setzte er seit FileMaker II ein, erst für eigene Zwecke, dann auch für andere Firmen und fertige Applikationen auf Runtime-Basis.
r.hermann@hr-softcom.de

Link macOS MBS-Plugin

Scannen mit FileMaker

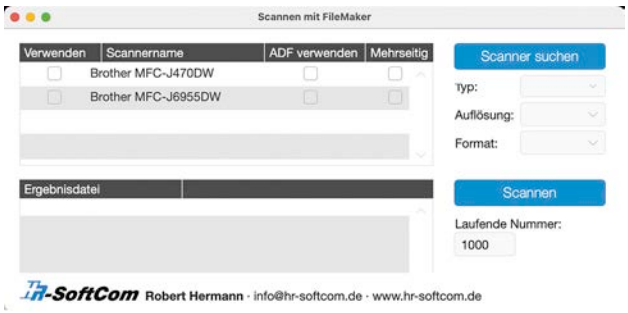
Alle Schritte in einer App

Vor ein paar Jahren fragte mich ein Kunde, ob es möglich wäre, direkt aus FileMaker zu scannen. Bis dahin war er folgendermaßen vorgegangen: Er hat das Dokument mit der Scan-App gescannt und das PDF dann in den Container gelegt. Hierzu musste er immer wieder die App wechseln – ein zusätzlicher Arbeitsschritt, den er gern einsparen wollte. Also machte ich mich daran einen Weg zu finden, um direkt aus FileMaker zu scannen.

Geholfen hat mir dabei das **MBS-Plugin**, das ich gern und häufig einsetze. Beim Schreiben dieses Artikels habe ich die Version 12.5 verwendet.

Im Hauptlayout der Beispieldatei werden in einem Portal die vorhandenen Scanner angezeigt und in einem weiteren Portal die gescannten Dateien. Zur Steuerung des Scanners gibt es die beiden Tasten „Scanner suchen“ und „Scannen“, drei Markierungsfelder namens **Verwenden**, **ADF verwenden**, **Mehrseitig** sowie die vier Einblendmenüs **Typ** („SW“ oder „Farbe“), **Auflösung, Format** und **Duplex**. Letzteres wird nur dann angezeigt, wenn der Scanner tatsächlich beide Seiten eines Blattes automatisch scannen kann.

Mit einem Klick auf die erste Taste wird der Scanner gesucht, die zweite Taste löst den Scan-Vorgang aus. Im unteren Teil des Layouts werden die gescannten Dateien gespeichert:

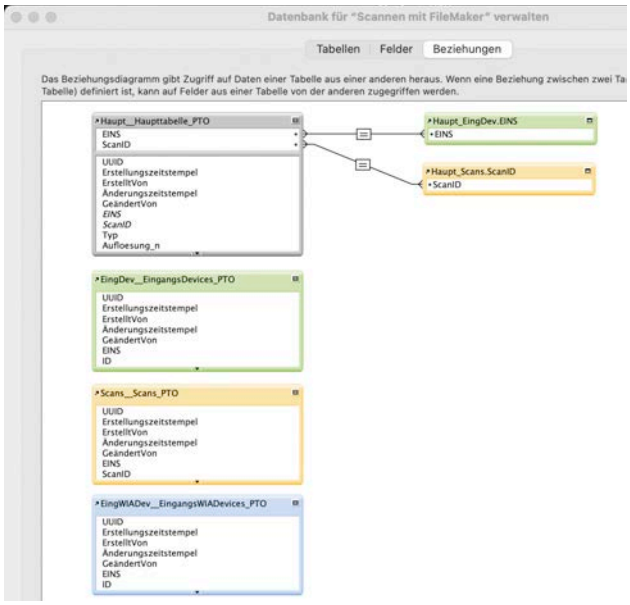


Tabellen

Zur Umsetzung der Scan-Funktion werden mehrere Tabellen benötigt – eine Haupttabelle namens ***Scannen mit FileMaker*** und drei Hilfstabellen: ***Scans*** dient zur Ablage der Bilder und in ***EingangsDevices*** bzw. ***EingangsWIADevices*** werden die Informationen der verwendeten Scanner am Mac bzw. am PC hinterlegt. In diesem Artikel beschäftige ich mich mit der macOS-Version, aber bei Interesse kann ich auch eine Anleitung für die Windows-Version erstellen, die dann in einer späteren Ausgabe des FileMaker Magazins erscheinen wird.

Tabellenauftreten

Für den gesamten Vorgang müssen nur zwei Beziehungen definiert werden, die beide zur Haupttabelle führen.



Felder

Da ich die Beispieldatei aus einer bestehenden Lösung extrahiert habe, sind einige Felder vorhanden, die nicht benötigt werden, aber auch nicht stören, weshalb ich sie nicht gelöscht habe. Zudem sind die Namen einiger Felder in Englisch, was dem **MBS-Plugin** geschuldet ist. Wer sich daran stört, kann diese Namen jederzeit ins Deutsche übersetzen.

Die Feldbenennung verwendet die in meiner Firma übliche Nomenklatur: Nach einem Unterstrich folgt die Information, um welche Art von Feld es sich handelt. Dabei steht „_n" für ein Zahlenfeld (**N**umber), „_g" für ein Globalfeld (**G**lobal) und „_c" für ein Formelfeld (**C**alculation). Möglich ist auch eine Kombination, z. B. „_ng" für ein globales Zahlenfeld oder „_nc" für ein Zahlenfeld mit einer Formel. Ansonsten sollten die Felder selbsterklärend sein, weshalb ich nicht weiter darauf eingehen werde.

Scripts

Interessanter sind die Scripts, die ich so unterteilt habe, dass man möglichst den Überblick behält.

Im Startscript wird das Fenster positioniert und die Datensätze in den verschiedenen Tabellen werden zur Vorbereitung geleert. Zudem wird in der Haupttabelle ein neuer Datensatz angelegt.

Das Script „Scanner suchen"

Mithilfe dieses Scripts werden die verfügbaren Scanner gesucht. Dabei werden sowohl die lokal am Mac z. B. per USB angeschlossenen Scanner berücksichtigt, als auch über das Netzwerk freigegebene Scanner. Hierfür wird die „ImageCapture"-Funktion des **MBS-Plugins** verwendet. Mit den folgenden Befehlen wird die **MBS**-Funktion aktiviert und die IDs der Scanner in Erfahrung gebracht:

```
◆ Variable setzen
  [ $r ; Wert: MBS ( "ImageCapture.Initialize" ) ]
◆ Variable setzen
  [ $Devices ; Wert: MBS ( ( "ImageCapture.Devices" ) ]
```

Über die Funktion „ImageCapture.DeviceInfo" werden verschiedene Parameter ausgelesen, z. B. die ID und der Name des Scanners, die Information, ob der Scanner remote oder direkt angeschlossen ist, sowie die Schnittstelle, über die der Scanner verbunden ist.

Um bestimmte Funktionen auslesen zu können, muss der Scanner aktiviert sein, was mithilfe von „ImageCapture.OpenDevice" erfolgt. Anschließend kann man z. B. die Funktion „ImageCapture.GetParameters" verwenden, die ein JSON mit allen möglichen Daten zurückgibt.

Zum Scannen sollte man wissen, welche Auflösungen mit dem Scanner möglich sind, ob er schwarz-weiß (SW) oder Farbe scannen kann und ob eine Duplex-Funktion verfügbar ist. Die Abfrage dieser Informationen erfolgt ebenfalls über die „ImageCapture.GetParameters"-Funktion, anschließend werden die Daten in die entsprechenden Felder geschrieben. Der Einfachheit halber habe ich auf die verschiedenen Formate verzichtet und mich auf Standardformate wie „DIN A4" usw. beschränkt.

Mit der Funktion „ImageCapture.CloseDevice" wird der Scanner geschlossen. Falls man nur einen Scanner zur Auswahl hat, kann man diesen gleich verwenden. Stehen mehrere Scanner zur Verfügung, überlassen wir die Auswahl dem Anwender.

Nachfolgend sehen Sie das dazugehörige Script in gekürzter Form. Aus Platzgründen werden alle Scripts im Artikel nur auszugsweise mit den wichtigsten Daten dargestellt. Die vollständigen Scripts finden Sie in der Beispieldatei.

`Mac - Scanner suchen | Fertig`

```
◆ Variable setzen
  [ $r ; Wert: MBS ( "ImageCapture.Initialize" ) ]
◆ # Scanner suchen
◆ Variable setzen
  [ $Devices ; Wert: MBS ( "ImageCapture.Devices" ) ]
◆ # Scanner prüfen
◆ Variable setzen
  [ $ID ; Wert: HoleWert ( $Devices ; $index ) ]
```

```
◆ Neuer Datensatz/Abfrage
◆ # Hier holen wir einzelne Informationen des Scanners
◆ Feldwert setzen
  [ EingDev__EingangsDevices_PTO::Name ;
  MBS ( "ImageCapture.DeviceInfo" ; $ID ; "name" ) ]
◆ Feldwert setzen
  [ EingDev__EingangsDevices_PTO::ID ; $ID ]
◆ #
◆ # Scanner öffnen
◆ Variable setzen
  [ $r ; Wert: MBS ( "ImageCapture.OpenDevice" ;
  EingDev__EingangsDevices_PTO::Name ) ]
◆ #
◆ # Wir lesen die Funktionalität des Scanners aus
◆ Feldwert setzen
  [ EingDev__EingangsDevices_PTO::Parameters ;
  JSONFormatElements ( MBS ( "ImageCapture.GetParameters" ) ) ]
◆ #
◆ # Wir schreiben die Daten für die Auflösung, für den aktiven
    Typ und ob der Scanner Duplex kann in Felder danach
    schließen wir den Scanner
◆ Wenn
  [ $r = "OK" ]
◆ Variable setzen
  [ $r ; Wert: MBS ( "ImageCapture.GetParameter" ;
  "availableFunctionalUnitTypes" ) ]
◆ Feldwert setzen
  [ EingDev__EingangsDevices_PTO::Resolutions_n ;
  MBS ( "ImageCapture.GetParameter" ; "supportedResolutions" ) ]
◆ Feldwert setzen
  [ EingDev__EingangsDevices_PTO::type ;
  MBS ( "ImageCapture.GetParameter" ; "pixelDataType" ) ]
◆ #
◆ Variable setzen
  [ $r ; Wert: MBS ( "ImageCapture.requestSelectFunctionalUnit" ;
  "DocumentFeeder" ) ]
◆ Feldwert setzen
  [ EingDev__EingangsDevices_PTO::Duplex ;
  MBS ( "ImageCapture.GetParameter" ; "supportsDuplexScanning" ) ]
◆ Ende (wenn)
◆ Variable setzen
  [ $r ; Wert: MBS ( "ImageCapture.CloseDevice" ) ]
```

Das Script „Scanner auswählen“

Wenn der Anwender das Kennzeichen „Verwenden“ anklickt, wird das Script „Scanner auswählen“ über einen Script-Trigger gestartet. Dieses Script wählt den Scanner aus und setzt die entsprechenden Daten in die Felder **Typ**, **Auflösung**, **Format** und **Duplex**. Klickt der Anwender erneut auf das Markierungsfeld, werden die Felder wieder geleert.

`Mac - Scanner auswählen ST | Fertig`

```
◆ Variable setzen
◆ # Ein Scanner ist ausgewählt worden
◆ Feldwert setzen
  [ Haupt_EingDev.EINS::Ausgewaehlt ; "1" ]
◆ # Scanner in die Globale schreiben
◆ Variable setzen
  [ $$device ; Wert: Haupt_EingDev.EINS::Name ]
◆ Feldwert setzen
  [ Haupt__Haupttabelle_PTO::Typ ; Haupt_EingDev.EINS::type ]
◆ # Als Standardwert für die Scanauflösung nehmen wir den
    zweitkleinsten Wert
◆ Feldwert setzen
  [ Haupt__Haupttabelle_PTO::Aufloesung_n ; Austauschen
  ( ElementeMitte ( Haupt_EingDev.EINS::Resolutions_n ; 2 ;
  1 ) ; "¶" ; "" ) ]
◆ # In das Dropdown-Menü kopieren wir die komplette
    Auflösungsliste des Scanners
◆ Feldwert setzen
  [ Haupt__Haupttabelle_PTO::ScannerAufloesung ;
  Haupt_EingDev.EINS::Resolutions_n ]
◆ # Als Standard legen wir A4 Hochformat fest
◆ Feldwert setzen
  [ Haupt__Haupttabelle_PTO::Format ; "A4 Hoch" ]
◆ # Als Standard für den Duplexscan geben wir "Nein" an
◆ Feldwert setzen
  [ Haupt__Haupttabelle_PTO::Duplex ; "Nein" ]
◆ Schreibe Änderung Datens./Abfrage
  [ Dateneingabeüberprüfung unterdrücken ; Mit Dialog: Ein ;
  Schreiben erzwingen ]
```

ADF verwenden

Die Abkürzung „ADF“ steht für den Dokumenteneinzug (Abk. für engl. Automatic Document Feeder). Beim Klick auf das Markierungsfeld „ADF verwenden“ sorgt ein Script-Trigger für den Aufruf des Scripts „Mac-ADF verwenden ST | Fertig“. Dieses Script bewirkt, dass die Auflösung und das Format in die entsprechenden Felder geschrieben werden und – falls der Scanner Duplex kann – die Duplexfunktion eingeblendet und mit „Nein“ belegt wird. Beim erneuten Klick wird die Duplexfunktion wieder ausgeblendet und Auflösung und Format entfernt. Ich zeige die Zeilen 33 bis 48 des gekürzten Scripts. Der erste Teil entspricht dem vorigen Script, deshalb lasse ich ihn hier weg:

`Mac - ADF verwenden ST | Fertig`

```
33 # "0" ist definitiv ohne Duplex-Scan, "1" ist sicher Duplex.
     "[MBS]-Fehlermeldung" könnte Duplex-Scan sein, deshalb
     blenden wir auf jeden Fall Duplex ein.
34 Wenn
   [ ZeichenLinks ( Haupt_EingDev.EINS::Duplex ; 5 ) = "[MBS]" ]
35 # Das ist die Dialogmeldung beim Scannen
36 Variable setzen
   [ $p ; Wert: MBS ( "ProgressDialog.SetTopText" ;
   "Scan vom ADF-Scanner." ) ]
37 # Duplexfunktion einblenden
38 Feldwert setzen
   [ Haupt__Haupttabelle_PTO::ScannerDuplex ;
   Falls ( Haupt_EingDev.EINS::Duplex = 0 ; "" ; 1 ) ]
39 Sonst, wenn
   [ Haupt_EingDev.EINS::Duplex = 1 ]
40 # Duplexfunktion einblenden
41 Feldwert setzen
   [ Haupt__Haupttabelle_PTO::ScannerDuplex ; 1 ]
42 Ende (wenn)
43 Schreibe Änderung Datens./Abfrage
   [ Dateneingabeüberprüfung unterdrücken ; Mit Dialog: Ein ;
   Schreiben erzwingen ]
```

```
44 Sonst
45 # ADF wird abgewählt
46 Variable setzen
   [ $$ADF ; Wert: "" ]
47 Feldwert setzen
   [ Haupt__Haupttabelle_PTO::ScannerDuplex ; "" ]
48 Ende (wenn)
```

Das Script „Mehrseitige Rechnung“

Hier wird lediglich die globale Variable **$$Mehrseitig** mit dem Wert „1“ befüllt oder geleert.

Das Script „Image to PDF using PDFKit“

Dieses Script ist zwar für das Scannen nicht notwendig, aber spätestens bei mehrseitigen Scans ist es sinnvoll, die Bilddateien in das PDF-Format zu konvertieren, damit alle Seiten in ein PDF-Dokument zusammengeführt werden können.

- Im ersten Schritt werden die Pfade zum Schreibtisch erstellt und der Ordner „Scantestordner“ wird definiert. Das erfolgt einmal für das **MBS-Plugin** und einmal für FileMaker selbst, denn das **MBS-Plugin** baut den Pfad anders auf als FileMaker.
- Anschließend (ab Zeile 32) wird mithilfe des **MBS-Plugins** ein neues PDF-Dokument erstellt, in welches die Bilder geladen werden. Das PDF-Dokument wird dann im Feld **ContainerPDF1** abgelegt.
- Im dritten Teil (ab Zeile 61) dieses Scripts wird ein weiterer Scan in ein neues PDF-Dokument geladen und im Feld **ContainerPDF2** gespeichert.
- Der vierte Teil (ab Zeile 70) sorgt für die Zusammenführung der beiden Dateien: Das PDF-Dokument aus dem zweiten Container wird in das PDF-Dokument aus dem ersten Container eingefügt. Hier das gekürzte Script:

`Image to PDF using PDFKit`

```
◆ # Erstellen eines neuen leeren PDF-Dokuments in einer
    Referenz: $ref
◆ Variable setzen
  [ $ref ; Wert: MBS ( "PDFKit.NewPDFDocument" ) ]
◆ #
◆ Variable setzen
  [ $r ; Wert: MBS ( "Files.CreateDirectory" ; $MBSPfad ) ]
◆ Wenn
  [ Scans__Scans_PTO::ContainerPDF2 = “” ]
◆ # Schleifenzähler auf 1
◆ Variable setzen
  [ $n ; Wert: 1 ]
◆ Schleife (Anfang)
◆ # Abrufen des Wertes $n in der Dateiliste
◆ Variable setzen
  [ $FileName ; Wert: HoleWert ( $Dateiname ; $n ) ]
◆ Verlasse Schleife wenn
  [ $FileName = "" ]
◆ # Prüfen ob es sich um eine JPEG-Datei handelt und nicht
    um eine Dot-Datei in Mac oder Windows (andere Formate
    können gelesen werden)
◆ Variable setzen
  [ $path ; Wert: MBS ( "Path.AddPathComponent" ; $MBSPfad ;
  $FileName ) ]
◆ Variable setzen
  [ $Image ; Wert: MBS ( "Files.ReadJPEG" ; $path ) ]
◆ Variable setzen
  [ $Error ; Wert: MBS ( "PDFKit.AddImagePage" ; $ref ;
  $Image ) ]
◆ # Beenden der Schleife, wenn das letzte Bild vorliegt
◆ Variable setzen
  [ $n ; Wert: $n+1 ]
◆ Schleife (Ende)
◆ Ende (wenn)
◆ #
◆ # Verwenden der $ref, um eine PDF-Datei im Container
  ContainerPDF1 zu erstellen, wenn es ein mehrseitiges PDF
  ist, werden die nächsten Seiten mit diesem PDF zusammengeführt
◆ # Erste Seite
◆ Wenn
  [ $$FirstPage = "" ]
◆ Feldwert setzen
  [ Scans__Scans_PTO::ContainerPDF1 ;
  MBS ( "PDFKit.GetPDFDocument" ; $ref ) ]
◆ Variable setzen
  [ $PDF ; Wert: MBS ( "PDFKit.OpenContainer" ;
  Scans__Scans_PTO::ContainerPDF1 ) ]
◆ Feldwert setzen
  [ Scans__Scans_PTO::ContainerPDF1 ; "" ]
◆ Feldwert setzen
  [ Scans__Scans_PTO::ContainerPDF1 ;
  MBS ( "PDFKit.Combine" ; $DateinamePDF ; $PDF ) ]
◆ #
◆ # Weitere Seiten
◆ Sonst
◆ # Verwenden der $ref, um eine PDF-Datei im Container
    ContainerPDF2 zu erstellen
◆ Feldwert setzen
  [ Scans__Scans_PTO::ContainerPDF2 ;
  MBS ( "PDFKit.GetPDFDocument" ; $ref ) ]
◆ Variable setzen
  [ $PDF ; Wert: MBS ( "PDFKit.OpenContainer" ;
  Scans__Scans_PTO::ContainerPDF2 ) ]
◆ Feldwert setzen
  [ Scans__Scans_PTO::ContainerPDF2 ; "" ]
◆ Feldwert setzen
  [ Scans__Scans_PTO::ContainerPDF2 ; MBS ( "PDFKit.Combine" ;
  $DateinamePDF ; $PDF ) ]
◆ Ende (wenn)
◆ #
◆ # PDF in ContainerPDF2 mit PDF in ContainerPDF1 zusammenführen
◆ Wenn
  [ Scans__Scans_PTO::ContainerPDF2 ≠ "" ]
◆ # Start mit leerer Liste
◆ Variable setzen
  [ $list ; Wert: "" ]
◆ # Laden der PDF-Datei aus dem Container
◆ Variable setzen
  [ $destPage ; Wert: 1 ]
◆ Variable setzen
  [ $r ; Wert: MBS ( "PDFKit.OpenContainer" ;
  Scans__Scans_PTO::ContainerPDF1 ) ]
```

```
◆ Variable setzen
  [ $list ; Wert: $list & $r & ¶ ]
◆ Variable setzen
  [ $r ; Wert: MBS ( "PDFKit.OpenContainer" ;
  Scans__Scans_PTO::ContainerPDF2 ) ]
◆ Variable setzen
  [ $list ; Wert: $list & $r & ¶ ]
◆ Feldwert setzen
  [ Scans__Scans_PTO::ContainerPDF1 ; MBS ( "PDFKit.Combine" ;
  Scans__Scans_PTO::LaufendeNr & ".pdf" ; $list ) ]
◆ Variable setzen
  [ $r ; Wert: MBS ( "PDFKit.Release" ; $list ) ]
◆ Ende (wenn)
◆ Feldwert setzen
  [ Scans__Scans_PTO::ContainerPDF2 ; "" ]
◆ #
◆ # Löschen des Speichers $ref
◆ Feldwert setzen
  [ Scans__Scans_PTO::ContainerJPEG ; "" ]
◆ Variable setzen
  [ $result ; Wert: MBS ( "PDFKit.Release" ; $ref ) ]
```

Das Script „Scannen“

Nachdem alle Vorarbeiten abgeschlossen sind, können wir jetzt endlich scannen. Das dazugehörige Script habe ich der Übersichtlichkeit halber in neun Schritte unterteilt.

- Im ersten Teil werden der Typ, die Auflösung, die Breite, die Höhe sowie die Duplex-Information in lokale Variablen geschrieben. Dann wird geprüft, welche Scanner verfügbar sind und der richtige ausgewählt. Wenn kein Scanner ausgewählt wurde, erscheint eine Fehlermeldung.
- Im zweiten Teil wird die Scan-Funktion initialisiert und die Verbindung zum Scanner geöffnet. Falls es einen Fehler gibt, wird dieser dem Anwender in einem Dialogfenster angezeigt.
- Anschließend werden die definierten Werte für den Scan gesetzt: das Papierformat, der Name des Scans, das Bildformat und der Ordner, in den die Scans gespeichert werden. Dann werden folgende Informationen abgefragt: Wird der Flachbettscanner oder der ADF verwendet, ist es ein Duplex-Scan, welches Papierformat (Maximal oder z. B. A4) wird angewendet und soll der Scan in SW oder in Farbe erfolgen. Die abgefragten Parameter setzen wir in einer Wenn-Sonst-Schleife mit der **MBS**-Funktion „ImageCapture.SetParameter“.
- Im vierten Teil wird der Scan-Dialog geöffnet und es erfolgt die Durchführung des/der Scans, dessen Ergebnis in dem temporären Ordner des Macs gespeichert wird.
- Im fünften und sechsten Teil werden evtl. Fehler mit einem Dialogfenster angezeigt, der Pfad zu den Bildern wird gesetzt und die Anzahl der Scans wird geholt.
- Der siebte Teil verarbeitet die einseitigen Scans, die in einer Schleife vom Bildformat in das PDF-Format konvertiert werden.
- Im achten Teil kümmert sich das Script um die mehrseitigen Scans. In einer Schleife wird die erste Seite verarbeitet, was nahezu identisch zum siebten Teil erfolgt. Dann gibt es eine weitere Schleife, in der die Folgescans verarbeitet und zur ersten Seite hinzugefügt werden.
- Im letzten Teil wird der Scanner geschlossen und wir kehren zum Hauptlayout zurück, wo die fertigen Scans im unteren Portal angezeigt werden.

Wir drucken hier nur die Teile 3 und 4 (Zeilen 75 bis 134 in der Beispieldatei) des Scripts ab, in denen der eigentliche Scan-Vorgang durchgeführt wird. Die anderen Teile können Sie in der Beispieldatei selbst analysieren.

`Mac - Scannen | Fertig`

```
◆ # ######################## Teil 3 ########################
◆ #
◆ # Parameter für den Scanner einstellen
◆ Variable setzen
  [ $r ; Wert: MBS ( "ImageCapture.SetParameter" ;
  "documentType" ; "A4" ) ]
◆ Variable setzen
  [ $r ; Wert: MBS ( "ImageCapture.SetParameter" ;
  "documentName" ; "Scan" ) ]
◆ Variable setzen
  [ $r ; Wert: MBS ( "ImageCapture.SetParameter" ;
  "documentUTI" ; "jpeg" ) ]
◆ Variable setzen
  [ $r ; Wert: MBS ( "ImageCapture.SetParameter" ;
  "downloadsDirectory" ; "temp" ) ]
◆ Variable setzen
  [ $r ; Wert: MBS ( "ImageCapture.requestSelectFunctionalUnit" ;
  "DocumentFeeder" ) ]
◆ Variable setzen
  [ $p ; Wert: MBS ( "ProgressDialog.SetTopText" ;
  "Scan vom Flachbettscanner." ) ]
◆ Variable setzen
  [ $r ; Wert: MBS ( "ImageCapture.SetParameter" ;
  "duplexScanningEnabled" ; 0 ) ]
◆ # Jetzt legen wir das Scann-Format fest
◆ Wenn
  [ $FormatX = "Maximal" ]
◆ Variable setzen
  [ $r ; Wert: MBS ( "ImageCapture.SetParameter" ;
  "scanArea" ; 0 ; 0 ; $physicalWidth ; $physicalHeight ) ]
◆ Sonst
◆ Variable setzen
  [ $r ; Wert: MBS ( "ImageCapture.SetParameter" ;
  "scanArea" ; 0 ; 0 ; $FormatX ; $FormatY ) ]
◆ Ende (wenn)
◆ # jetzt schauen wir, ob wir farbig oder SW scannen wollen
◆ Wenn
  [ $Typ = "SW" ]
◆ Variable setzen
  [ $r ; Wert: MBS ( "ImageCapture.SetParameter" ;
  "bitDepth" ; 8 ) ]
◆ Variable setzen
  [ $r ; Wert: MBS ( "ImageCapture.SetParameter" ;
  "pixelDataType" ; "Gray" ) ]
◆ Sonst
◆ Variable setzen
  [ $r ; Wert: MBS ( "ImageCapture.SetParameter" ;
  "bitDepth" ; 8 ) ]
```

```
◆ Variable setzen
  [ $r ; Wert: MBS ( "ImageCapture.SetParameter" ;
  "pixelDataType" ; "RGB" ) ]
◆ Ende (wenn)
◆ Variable setzen
  [ $r ; Wert: MBS ( "ImageCapture.SetParameter" ;
  "resolution" ; $Resolution ) ]
◆ #
◆ # ######################## Teil 4 ########################
◆ #
◆ # Hier wird der Scann-Dialog aktiviert und die
    entsprechenden Texte eingefügt
◆ Variable setzen
  [ $p ; Wert: MBS ( "ProgressDialog.Reset" ) ]
◆ Variable setzen
  [ $p ; Wert: MBS ( "ProgressDialog.SetTitle" ; "Scannen" ) ]
◆ Variable setzen
  [ $p ; Wert: MBS ( "ProgressDialog.SetBottomText" ; "" ) ]
◆ Wenn
  [ $$ADF ≠ "" ]
◆ Variable setzen
  [ $p ; Wert: MBS ( "ProgressDialog.SetTopText" ;
  "Scan vom ADF-Scanner." ) ]
◆ Sonst
◆ Variable setzen
  [ $p ; Wert: MBS ( "ProgressDialog.SetTopText" ;
  "Scan vom Flachbettscanner." ) ]
◆ Ende (wenn)
◆ Variable setzen
  [ $p ; Wert: MBS ( "ProgressDialog.SetButtonCaption" ;
◆ "Abbrechen" ) ]
◆ Variable setzen
  [ $p ; Wert: MBS ( "ProgressDialog.SetProgress" ; -1 ) ]
◆ Variable setzen
  [ $p ; Wert: MBS ( "ProgressDialog.Show" ) ]
◆ # Dann scannen wir mal
◆ Variable setzen
  [ $r ; Wert: MBS ( "ImageCapture.requestScan" ) ]
◆ Variable setzen
  [ $text ; Wert: ZeichenMitte ( $r ; 7 ; 999 ) ]
◆ Variable setzen
  [ $p ; Wert: MBS ( "ProgressDialog.Hide" ) ]
```

Ich habe mir eine Datei mit dieser Funktion zunächst für meine Eingangsrechnungen programmiert, verwende sie aber inzwischen auch in anderen Bereichen, z. B. für das Dokumentenmanagement. Es ist einfach sehr praktisch, wenn man die Datenbank zum Scannen von Dokumenten nicht verlassen muss. Es gibt sicher noch viele weitere Möglichkeiten, wo Sie die gezeigten Möglichkeiten dieser Programmierung einsetzen können. Falls noch Fragen offen sind, können Sie sich gerne an mich wenden. Auch über Verbesserungsvorschläge würde ich mich freuen. ■

FileMaker Magazin

Das FileMaker Magazin

- Einzige deutschsprachige Fachzeitschrift zu FileMaker
- Wissen aus erster Hand von anerkannten FileMaker Fachautoren
- Große Themenvielfalt für Anwender und Entwickler

Exklusiv für Premium-Abonnenten

- Sechs FMM Ausgaben pro Jahr
- Kostenlose Nutzung des Abonnentenbereichs auf ***www.filemaker-magazin.de***
- PDF-Archiv mit allen bisher veröffentlichen Ausgaben
- Jede Ausgabe mit kostenlosen Beispieldateien und Zusatzinfos zum Download

Unser Service

- Aktuelle Neuheiten, Tipps und Infos, Kleinanzeigen und vieles mehr jederzeit auf unseren Webseiten
- Hilfe bei allen Fragen zu FileMaker im FMM Forum
- Kompetente Beratung zum Kauf von FileMaker Lizenzen: Einfach anrufen +49 (0)40 589 65 79 70.

Wenn Sie sich für ein FileMaker Magazin **Abo** interessieren, klicken Sie bitte hier!

Eine kostenlose **Leseprobe** des FileMaker Magazins erhalten Sie, wenn Sie hier klicken.

Hier finden Sie Aktuelles zu **FileMaker Lizenzen**, egal ob Sie kaufen, mieten oder sich einfach informieren möchten.