# ONLINEBANKING MIT ACCESS



# BANKGESCHÄFTE ERLEDIGEN MIT ACCESS UND DER DDBAC-KOMPONENTE



ANDRÉ MINHORST

André Minhorst

# Onlinebanking mit Access

### Bankgeschäfte erledigen mit Microsoft Access und der DDBAC-Komponente

Vorabversion für Buchbesteller. Stand: 29. September 2014

#### André Minhorst – Onlinebanking mit Access

#### ISBN 978-3-944216-02-7

© 2014 André Minhorst Verlag, Borkhofer Straße 17, 47137 Duisburg/Deutschland

1. Auflage 2014

Lektorat André Minhorst Korrektur Rita Klingenstein Cover/Titelbild André Minhorst Typographie, Layout und Satz André Minhorst Herstellung André Minhorst Druck und Bindung Kösel, Krugzell (www.koeselbuch.de)

#### Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliographie. Detaillierte bibliografische Daten finden Sie im Internet unter http://dnb.d-nb.de.

Das vorliegende Werk ist in all seinen Teilen urheberrechtlich geschützt. Alle Rechte vorbehalten, insbesondere das Recht der Übersetzung, des Vortrags, der Reproduktion, der Vervielfältigung auf fotomechanischem oder anderen Wegen und der Speicherung in elektronischen Medien. Ungeachtet der Sorgfalt, die auf die Erstellung von Text, Abbildungen und Programmen verwendet wurde, können weder Verlag noch Autor, Herausgeber oder Übersetzer für mögliche Fehler und deren Folgen eine juristische Verantwortung oder irgendeine Haftung übernehmen.

Die in diesem Werk wiedergegebenen Gebrauchsnamen, Handelsnamen, Warenbezeichnungen et cetera können auch ohne besondere Kennzeichnung Marken sein und als solche den gesetzlichen Bestimmungen unterliegen.

Die in den Beispielen verwendeten Namen von Firmen, Produkten, Personen oder E-Mail-Adressen sind frei erfunden, soweit nichts anderes angegeben ist. Jede Ähnlichkeit mit tatsächlichen Firmen, Produkten, Personen oder E-Mail-Adressen ist rein zufällig.

Keiner fühlt's wie wir.

Vorwo	rt	11	
1	Installation	15	
1.1	Voraussetzungen		
1.2	Lizenzen		
1.3	Installation ohne SDK		
2	Einrichtung	17	
2.1	Homebanking-Kontakt erstellen		
2.2	Konfiguration bearbeiten		
2.2.1	Kontakt-Informationen		
2.2.2	PIN- und TAN-Funktionen		
2.2.3	TAN-Informationen abfragen		
2.2.4	TAN-Verfahren auswählen		
2.3	Speicherort der Konfiguration		
2.4	Debugging/Protokolldatei		
2.5	Bankverbindung zum Testen der Anwendung		
3	Grundlagen der DDBAC	41	
3.1	VBA-Zugriff		
3.2	Objektmodell der DDBAC		
3.3	Das BACBanking-Objekt		
3.3.1	Einfacher Zugriff auf das BACBanking-Objekt		
3.3.2	Kontaktverwaltung öffnen		
3.3.3	Neuen Kontakt einrichten		
3.3.4	Die Customers-Auflistung		
3.3.5	BACCustomer löschen		
3.3.6	BACCustomer anlegen		
3.3.7	Customer synchronisieren		
3.3.8	Optionen		
3.3.9	Optionen von Access aus einstellen		
3.3.10	BPD und UPD		
3.4	Das BACCustomer-Objekt		
3.5	Das BACContact-Objekt		
3.5.1	Zugriff auf BACContact-Objekte		
3.5.2	Vereinfachung für den Zugriff auf die BACContacts-Auflistung		
3.5.3	BACContact-Fields		
3.5.4	Die AccountData-Eigenschaft		
3.5.5	Die BankData-Eigenschaft		
3.5.6	Die Accounts-Auflistung		
3.5.7	BACContact-Objekte per Kombinationsfeld		
3.5.8	Dialoge für die Verwaltung eines Kontakts		
3.5.9	Weitere Funktionen und Methoden von BACContact		

3.6	Das BACAccount-Objekt	
3.6.1	Auf bestehende BACAccount-Objekte zugreifen	
3.6.2	Zugriff per VBA	
3.6.3	Eigenschaften des BACAccount-Objekts	
3.6.4	Die Segment-Eigenschaft	
3.6.5	Die FindOptimalBPDSegment-Funktion	
3.7	Das BACSegment-Objekt	
3.7.1	Segment erstellen	
3.7.2	Segment mit den Daten eines Accounts füllen	
3.7.3	HKSAL und andere aus HIUPD füllen	
3.7.4	BACSegment-Anwendung: Die Bankparameterdaten (BPD)	
3.8	Das BACDialog-Objekt	
3.8.1	BACDialog: Gültigkeit festlegen	
3.8.2	Einen BACDialog starten	
3.8.3	Segment während des Dialogs ausführen	
3.9	Das Objekt BACMessage	
3.9.1	Acknowledgement – Rückgabe des Ergebnisses	
3.9.2	Transactions – die eigentlichen Informationen	
3.9.3	Das Transaction-Objekt	
3.10	Debuggen	
3.10.1	Anfrage prüfen	
3.10.2	Antwort prüfen	
3.10.3	Fehlercodes	
3.10.4	Protokollierung	
3.10.5	Hinweise auf fehlerhafte Segmente	
3.10.6	Ausgabe formatieren	
3.11	Einfache Protokollierung	
3.12	Dokumente	
4	Konten per VBA verwalten	135
4.1	Kontakt herstellen	135
4.2	Bankverbindungen einlesen.	136
4.3	Letzten Contact und Account speichern	142
4.4	Konto-Informationen anzeigen	145
4.5	Dialog zum Verwalten der Konten per VBA öffnen	
4.6	Mögliche Transaktionen des Kontos ermitteln	
4.6.1	Ablauf der Prozedur	
4.6.2	Segmente und Eigenschaften einlesen	
4.6.3	Segmentformular erweitern	
4.6.4	Suchfunktion ergänzen	
4.6.5	Unterformular integrieren	
4.6.6	Erweiterung: Segment-Informationen anzeigen	
4.6.7	Nutzen des Formulars	
4.7	Transaktionen ausführen	
5	Kontostand ermitteln	173

5.1	Betroffenes Konto ermitteln	
5.2	Alle Konten einlesen	
5.2.1	Öffnen des Formulars	
5.2.2	Unterformular aktualisieren	
5.2.3	Aktualisieren der Konten nach Kontakt-Auswahl	
5.2.4	Kontostand aktualisieren	
5.2.5	Alle Konten einlesen, zuverlässige Variante	
5.3	Bank-Verbindung mit einmaliger Authentifizierung erhalten	
5.3.1	Flexiblere Lösung	
5.3.2	Einbau in die Funktion KontostandErmittelnKonto	
5.4	Umsatzübersicht per Bericht	
5.4.1	Datenherkunft des Berichts festlegen	
5.4.2	Gruppierungen definieren	
5.4.3	Nur bestimmte Konten anzeigen	
5.4.4	Konto aus- oder abwählen	
6	Umsätze einlesen	207
6.1	Ablauf beim Einlesen der Umsätze	
6.2	Einlesen der Umsätze im Detail	
6.3	Die Klasse clsTransactions	
6.4	Tabelle zum Speichern der Umsatzdaten	
6.4.1	Problem Abholzeitpunkt	
6.4.2	Klasse zum Sammeln der Umsatzpositionen	
6.5	Umsätze im Formular anzeigen	
6.5.1	Multikontenfähigkeit	
6.5.2	Details einer Umsatzposition anzeigen	
6.6	Umsätze mit Aufsetzpunkt	
6.7	Umsatzübersicht	
6.7.1	Fehlende Daten	
6.7.2	Kontoauszugsbericht erstellen	
6.7.3	Kontoauszug nach Datum	
7	SEPA-Überweisung	251
7.1	Überweisungsformular verwenden	
7.2	Überweisungsformular erstellen	
7.2.1	Kombinationsfelder füllen	
7.2.2	Das Kombinationsfeld zur Auswahl der Empfänger	
7.2.3	Empfänger bei Überweisung speichern	
7.2.4	Neuen Empfänger eingeben	
7.2.5	Empfänger auswählen	
7.3	Empfänger bearbeiten	
7.3.1	Unterformular sfmEmpfaenger erstellen	
7.3.2	Überweisungsdaten	
7.3.3	Zeichenanzahl und gültige Zeichen beim Verwendungszweck	
7.4	Überweisung anstoßen	
7.5	Einfache Überweisung ausführen	

7.5.1	Beispielüberweisung	
7.5.2	Antwort auf die Beispielüberweisung	
7.6	Überweisung als Vorlage speichern	
7.6.1	Vorlagen-Steuerelemente hinzufügen	
7.6.2	Tabelle zum Speichern der Vorlagen	
7.6.3	Kombinationsfeld zur Auswahl von Vorlagen füllen	
7.6.4	Vorlage speichern	
7.6.5	Vorlage auswählen	
7.7	Terminüberweisungen	
8	SEPA-Terminüberweisungen	281
8.1	Formular zum Ausführen von Terminüberweisungen	
8.2	Terminüberweisung ausführen	
8.3	Terminüberweisungen abrufen	
8.3.1	Terminüberweisungen verwalten	
8.3.2	Terminüberweisungen einlesen	
8.4	Terminüberweisung löschen	
8.4.1	Aufruf der Löschen-Funktion	
8.4.2	Funktion zum Löschen einer Terminüberweisung	
8.5	Fehlermeldungen ausgeben	
8.6	Terminüberweisung ändern	
9	SEPA-Daueraufträge	307
9.1	Informationen für das Anlegen eines Dauerauftrags	.308
9.1.1	Informationen vom Kreditinstitut holen	
9.1.2	Dauerauftrag-geeignete Banken ermitteln	
9.1.3	Informationen für die Datumseingabe von der Bank einlesen	
9.1.4	Basisdaten aktualisieren	
9.2	Dauerauftrag durchführen	
9.2.1	Dauerauftrag vorbereiten	
9.2.2	Dauerauftrag durchführen	
9.3	Bestand der Daueraufträge einlesen	
9.3.1	Tabelle zum Speichern der Daueraufträge	
9.3.2	Unterformular zum Anzeigen der Daueraufträge	
9.3.3	Formular zum Verwalten der Aufträge	
9.3.4	Abrufen der bestehenden Daueraufträge	
9.4	Daueraufträge löschen	
10	Lastschriften mit SEPA	353
10.1	Grundlagen zu Lastschriften	
10.1.1	SEPA-Basislastschrift	
10.1.2	SEPA-Firmenlastschrift	
10.1.3	International	
10.1.4	Schlüsselwörter	
10.1.5	Verschiedene Lastschriftarten	
1016	Finlieferung der Lastschrift	356

10.1.7	SEPA-Lastschriftmandat	
10.1.8	Benötigte Daten	
10.1.9	Nötige Informationen im Überblick	361
10.2	Basis-Lastschriften durchführen	362
10.2.1	Lastschrift starten	364
10.2.2	Durchführen der Einzellastschrift	366
10.3	Verwalten der Lastschriften	372
10.3.1	Sammellastschriften einlesen	375
10.3.2	Lastschrift löschen	378
10.4	Lastschrifttyp festlegen	
10.5	Datenmodell für SEPA-Lastschriften	
11	SEPA-Lastschriften verwalten	385
11.1	Umstellung vom Einzugsermächtigungsverfahren zum SEPA-Basis-Lastschriftverfahren.	
11.2	Kontonummer und BLZ konvertieren	
11.2.1	Beispiele für den Einsatz der DLL	
11.2.2	Testen der Funktionen	
11.3	Massenweise Konvertierung zu IBAN/BIC	
11.4	Umstellung auf SEPA-Lastschriften	403
11.4.1	Stammdaten des Empfängers speichern	403
11.4.2	Mandatsreferenznummer anlegen	404
11.4.3	Bericht zur Lastschriftumstellung erstellen	405
11.5	Sammellastschrift mit mehreren Aufträgen	
11.5.1	Wann erfolgt welche Lastschrift?	409
11.5.2	Mitgliedsdaten speichern	409
11.5.3	Mitgliedsbeiträge speichern	411
11.5.4	Formular zum Verwalten der Mitgliedsbeiträge.	411
11.5.5	Aktionen beim Öffnen des Formulars	
11.5.6	Filtern des Unterformulars sfmMitgliedsbeitraege	
11.5.7	Fällige Beiträge ermitteln	418
11.5.8	Die Auswahl benutzen	
11.5.9	Konto für den Auftraggeber auswählen	420
11.5.10	IBAN und BIC für Kontonummer und BLZ ermitteln	
11.5.11	Lastschrift vormerken	424
11.5.12	Lastschriften einreichen	
12	Fehlerbehandlung	435
12.1	Test-PIN und -TAN	
12.2	Fehlerbehandlungen	
12.2.1	PIN gesperrt.	436
12.2.2	PIN muss vor erstmaliger Anmeldung geändert werden.	438
12.2.3	Abbruch des Dialogs und Rückmeldung mit Code 9333	
12.2.4	Vorläufige PIN-Sperre	
12.2.5	Verarbeitungssystem nicht verfügbar	
12.2.6	PIN ungültig	
12.2.7	PIN gültig	

12.3	Tests mit der TAN	
12.3.1	TAN verwendet	
12.3.2	TAN-Liste gesperrt	
12.3.3	TAN ungültig (bei zwei-Schritt-TAN-Verfahren)	
13	Tools und Hilfsfunktionen	447
13.1	Die Funktion GetXMLElement	
13.2	Die Funktion FormatXML	
13.3	Die Funktion IsoDatum	
13.4	Die Funktion SQLDatum	
13.5	Die Funktion DatumAusXML	
13.6	Die Funktion IstFormularGeoeffnet	
13.7	Die Funktion XMLAnonymisieren	
14	Onlinebanking per Webservice	453
14.1	Funktionsweise des Webservice	
14.2	SOAP-Request ermitteln/erstellen	
14.2.1	www.soapclient.com	
14.2.2	SoapUI	
14.3	Mögliche Anfragen per Webservice	
14.4	BIC für eine BLZ ermitteln	
14.4.1	Ermittlung per VBA-Funktion	
14.4.2	Funktion zum Abfragen des BIC zu einer BLZ	
14.5	IBAN und BIC ermitteln	
14.6	BIC für IBAN ermitteln	
14.7	IBAN prüfen	
14.8	Bankinformationen abfragen	
14.9	Geschäftsvorfälle mit Authentifizierung	
14.10	Kontostand abrufen	
14.10.1	Tabelle zum Speichern der Kontakte	
14.10.2	Kontoinformationen einlesen	
14.10.3	Kombinationsfelder füllen	
14.10.4	Datenherkunft des zweiten Kombinationsfeldes	
14.10.5	ContactData und Konten ermitteln	
14.10.6	Webservice aufrufen	
14.10.7	Konten in die Tabelle tblKonten schreiben	
14.10.8	Aktualisieren des Kontakts	
14.10.9	Sicherheitsverfahren aktualisieren	
14.10.10	Geschäftsvorfälle aktualisieren	
14.11	Kontostand einlesen	
14.11.1	Die Funktion BalanceRequest	
14.12	Umsätze einlesen	
14.13	Überweisung durchführen	
14.13.1	Überweisung starten	
14.13.2	Response ohne gewünschte Antwort	

## Vorwort

Onlinebanking – da denken die meisten an die Online-Version ihrer Bank oder vielleicht auch an eine Software wie etwa von Lexware, die den Zugriff auf Konten, Kontostände oder Umsätze erlauben. Wenn es in Richtung Access geht, gibt es hier und da jemanden, der seine Lastschriften mithilfe einer per VBA generierten Datenträgeraustausch-Datei durchführt.

Alles Weitere überlassen wir dann besser den Experten, mag man sich an dieser Stelle denken. Aber mit dem geeigneten Werkzeug machen Sie auch Ihre Access-Anwendung Onlinebankingfähig: Die Firma *B* + *S* Banksysteme AG hat einen Satz von Komponenten entwickelt, die den Zugriff auf Ihre Kontostände und Umsätze erlaubt, die sogenannte *DDBAC*. Um diese Komponenten zu nutzen, brauchen Sie nur wenige Dinge zu erledigen:

- » Sie installieren die Komponenten auf dem jeweiligen Rechner.
- » Sie richten die Bankverbindungen und Konten ein.
- » Sie fügen Ihrer Anwendung Code hinzu, um Kontostände und Umsätze einzulesen und alle möglichen Transaktionen durchzuführen im einfachsten Fall eine Überweisung.

Die zusätzlich benötigte Software ist leider nicht kostenlos. Der Hersteller trägt, wie Sie sich vorstellen können, eine hohe Verantwortung, muss er doch sicherstellen, dass bei finanziellen Transaktionen keine Fehler unterlaufen. Aber im Rahmen dieses Buchs biete ich Lizenzen der Software zu einem für kleinere Mengen sehr günstigen Preis an (mehr dazu im Onlineshop unter *shop.minhorst.com*).

Wer die in diesem Buch beschriebenen Techniken nur für den privaten Einsatz nutzen möchte oder diese für den Einsatz in einer Access-Anwendung evaluieren möchte, kann das SDK kostenlos herunterladen und damit experimentieren.

Wenn Sie jedoch geschäftliche Transaktionen damit durchführen oder eigene Software um die DDBAC-Komponente erweitern möchten, müssen Sie entsprechende Lizenzen kaufen. Dies gilt auch, wenn Sie eigene Anwendungen mit der DDBAC ausstatten und diese an Kunden weiterverkaufen – für jede Kopie ist dann eine Lizenz erforderlich.

#### Besser als die Bank-Webseiten

Die Chancen stehen gut, dass Sie Bankgeschäfte online erledigen – wer so fit am Rechner ist, dass er sich mit Access beschäftigt, wird wohl auch nicht wegen jeder Überweisung zur Bankfiliale laufen. Was teilweise ja auch mittlerweile mit unverhältnismäßig hohem Aufwand verbunden ist – mein Bankinstitut hat beispielsweise die Filiale vor Ort geschlossen – die nächste Filiale ist nur noch mit entsprechenden Verkehrsmitteln zu erreichen. Wer Konten bei mehr als einem Kreditinstitut hat, weiß außerdem, dass man sich nicht mal eben auf die Schnelle einen Überblick über alle Kontostände und Umsätze verschaffen kann. Immerhin müssen Sie

#### Vorwort

für jedes Kreditinstitut eine eigene Webseite aufrufen, Ihre Zugangsdaten eingeben und die gewünschten Daten einsehen. Mit den in diesem Buch beschriebenen Techniken ist es einfach, beispielsweise alle Kontostände in einem Rutsch einzulesen und diese übersichtlich etwa per Bericht darzustellen.

Und während die Online-Übersicht der Kontoumsätze immer nur einen Überblick über die letzten 90 Tage verschafft, können Sie sich mit den Funktionen aus diesem Buch immer die letzten Umsätze in eine entsprechende Tabelle schreiben. Das gelingt zwar auch nur für einen Zeitraum von rund 90 Tagen, aber wenn Sie Ihre Umsätze alle zwei, drei Monate abrufen, haben Sie doch alle Umsätze auf dem Rechner und können Sie so komfortabel einen Überblick verschaffen.

#### Geschäftsvorfälle

Die Anzahl der in diesem Buch behandelten Geschäftsvorfälle ist überschaubar: Kontostände einlesen, Kontoumsätze erfassen, SEPA-Überweisungen tätigen (mit und ohne Termin), Daueraufträge einrichten und Lastschriften einreichen. Sie finden jedoch den kompletten Quellcode mit ausführlicher Dokumentation im Buch, sodass Sie weitere Geschäftsvorfälle, die letztlich alle ähnlich aufgebaut sind, leicht selbst hinzufügen können. Dafür finden Sie in einem umfangreichen Kapitel alle verwendeten Objekte mit ihren Eigenschaften, Methoden und Auflistungen.

Das Thema XML spielt hier eine große Rolle, denn die meisten Informationen, die Ihnen Ihr Kreditinstitut zurücksendet, kommen in diesem Format. Wir zeigen aber, wie Sie die gewünschten Informationen aus den XML-Dokumenten auslesen.

#### **Die DDBAC-Bibliothek**

Die in diesem Buch vorgestellten Techniken basieren auf den Objekten, Methoden und Eigenschaften der DDBAC-Komponenten der Firma *B+S Banksysteme AG*. Die Bibliothek können Sie für private Zwecke kostenfrei nutzen. Wenn Sie diese jedoch für geschäftliche Zwecke nutzen möchten (und wenn Sie auch nur Ihre Rechnungseingänge prüfen), benötigen Sie eine Lizenz. Diese erhalten Sie entweder direkt beim Hersteller oder in meinem Onlineshop unter *www. amvshop.de*. Für kleinere Stückzahlen kommen Sie in meinem Shop wahrscheinlich günstiger weg, sollten Sie eigene Produkte mit der DDBAC ausstatten und diese in größeren Mengen verkaufen wollen, ist die *B* + *S Banksysteme AG* möglicherweise Ihr Ansprechpartner. Bei diesbezüglichen Fragen stehe ich gern unter *andre@minhorst.com* zur Verfügung.

#### Testsystem

Für die Leser dieses Buchs stehen mit dem Zeitpunkt der Veröffentlichung zwei Testkonten bereit, mit denen Sie das Einlesen von Kontoständen und -umsätzen, Überweisungen, Daueraufträge, Lastschriften et cetera testen können, ohne echtes Geld hin- und herschieben zu müssen. Bitte gehen Sie verantwortungsvoll mit diesen Testkonten um, da diese bei unsachgemäßem Gebrauch leider deaktiviert werden müssen. Ich kenne die Leser meiner Veröffentlichungen je-

Vorwort

doch als rücksichtsvolle Menschen, bei denen ich mir diesbezüglich überhaupt keine Sorgen mache.

#### Beispieldatenbanken

Im Download zum Buch im Kundenkonto des Shops (Anmeldung erforderlich) finden Sie eine Zip-Datei mit allen notwendigen Dateien mit Ausnahme der DDBAC-Komponente.

Diese enthält zwei Datenbankdateien für Access 2003 und für Access 2007 und neuer. Der Unterschied ist, dass die Version ab Access 2007 mit einem schicken Ribbon und Icons in Formularen und auf Schaltflächen ausgestattet sind. Da Access 2003 demnächst nicht mehr unterstützt wird, habe ich den Fokus nicht mehr auf diese Version gelegt. Sie enthält dennoch alle Funktionen, die im Buch beschrieben werden und stellt die Formulare und Berichte über ein Übersichtsformular bereit.

#### Auf geht's!

Arbeiten Sie das Buch durch, es lohnt sich. Ich habe mich mehr als sechs Monate intensiv mit der Bibliothek und ihren Möglichkeiten beschäftigt und vieles herausgefunden, was Sie sich nun durch die Lektüre dieses Buches in kurzer Zeit aneignen können. Außerdem finden Sie bereits Beispiele für viele Geschäftsvorfälle, die Sie leicht für andere Geschäftsvorfälle, die hier nicht abgebildet sind, anpassen können.

Duisburg, 26. September 2014

André Minhorst

# **1** Installation

Wer ein echter Access-Entwickler ist, der gibt sich natürlich nicht mit einer Homebanking-Software von der Stange ab. Nein, richtige Cracks bauen sich ihren Geldverwalter selbst – und natürlich mit Access. Was, Sie haben keine Idee, wie Sie per VBA auf Ihre Konten zugreifen können? Kein Problem: Dafür gibt es komplette Funktionssammlungen und gar ganze Frameworks – zum Beispiel die für diese Lösung verwendeten *DataDesign Banking Application Components* (*DDBAC*) der Firma *B+S Banksysteme AG*.

Die benötigten Komponenten sind für den Privatgebrauch und für Testzwecke kostenlos. Erst wenn Sie eine Anwendung entwickeln, welche mit DDBAC-Komponenten zusammenarbeitet, und Sie diese vertreiben, fallen Lizenzkosten an (nähere Informationen über die Konditionen erhalten Sie beim Hersteller selbst). Das gilt ebenfalls, wenn Sie die Komponente für eigene geschäftliche Zwecke nutzen.

In diesem Dokument lernen Sie die grundlegenden Techniken für den Einsatz dieser Komponente per VBA kennen.

### 1.1 Voraussetzungen

Die vorliegende Lösung setzt die Installation der *DDBAC*-Komponenten der Firma *B* + *S* Banksysteme AG voraus. Diese können Sie in der aktuellen Fassung unter folgendem Link herunterladen: http://www.ddbac.de/Dev.SDK.shtml. Sie benötigen das DDBAC Software Development Kit in der jeweils aktuellen Version. Nach dem Herunterladen und dem Installieren wäre dieser Teil der Vorbereitungen schon abgehakt. Bitte vergessen Sie nicht, die Lizenzbedingungen sorgfältig zu lesen.

Eine weitere Voraussetzung ist ein Bankkonto, dessen HBCI-Funktionen freigeschaltet sind, und ein Satz von Informationen, die Sie für den Einsatz der DDBAC-Komponenten benötigen. Details hierzu finden Sie beim Kreditinstitut Ihres Vertrauens – aber wahrscheinlich sind Sie ohnehin schon längst online unterwegs, was Ihre Bankgeschäfte angeht.

Die Installation des DDBAC-Pakets hat eine ganze Reihe Dateien auf Ihre Festplatte geschaufelt, die Sie sich über das Startmenü von Windows zugänglich machen können. Dort finden Sie unter anderem eine Reihe Beispiele und haufenweise Dokumentation. Welche Informationen für Sie interessant sein können, zeigen wir Ihnen im Laufe dieses Kapitels. Kapitel 1 Installation

### 1.2 Lizenzen

Wenn Sie eine oder mehrere Lizenzen für den geschäftlichen Eigengebrauch oder für den Einbau in Lösungen für Ihre Kunden erwerben möchten, finden Sie diese im Onlineshop unter folgender Internetadresse: *www.amvshop.de*. Dort finden Sie auch die Lizenzbedingungen.

### 1.3 Installation ohne SDK

Wenn Sie die DDBAC-Komponente ohne SDK und nur für die reine Nutzung der Komponente installieren möchten, können Sie die *DDBAC.exe* verwenden. Diese finden Sie unter folgendem Link:

http://ddbac.de/index.php/menu-endanwender-user/menu-e-u-updates-download

Den dortigen Download der *DDBAC.EXE* binden Sie nach Wunsch in das Setup Ihrer Anwendung ein.

# 2 Einrichtung

Dieses Kapitel beschreibt die Einrichtung der Konten über die entsprechende Komponente der Systemsteuerung.

### 2.1 Homebanking-Kontakt erstellen

Für die Verwendung der DDBAC-Komponenten benötigen Sie einen sogenannten Homebanking-Kontakt, der einem Benutzerkonto bei einem Kreditinstitut entspricht. Das ist keinesfalls identisch mit einem Bankkonto – auch wenn es möglicherweise später den Anschein hat, weil die Nummern von Benutzerkonto und Bankkonto Ähnlichkeiten aufweisen oder gar gleich sind.

Das Benutzerkonto entspricht vielmehr dem, was Sie auch vorfinden, wenn Sie sich über die Internetseite Ihrer Bank einloggen, um dort etwa Kontostände abzurufen oder Überweisungen vorzunehmen. In vielen Fällen scheinen Benutzerkonto und Bankkonto identisch, wenn Sie nur ein Konto bei der Bank eingerichtet haben – etwa ein Girokonto.

Wenn Sie noch weitere Konten, Depots oder Sparkonten bei dieser Bank hätten, würden diese jedoch wahrscheinlich online über das gleiche Benutzerkonto zugänglich sein.

Und den Zugriff auf ein solches Benutzerkonto müssen Sie nun einrichten, wenn Sie mit Access auf Ihre Bankkonten zugreifen möchten. Die nötigen Daten erhalten Sie von Ihrem Kreditinstitut, den Rest erledigt der *Administrator für Homebanking Kontakte*. Den finden Sie nicht etwa im Startmenü bei den übrigen Dateien der DDBAC-Installation, sondern gut untergebracht in der Systemsteuerung (siehe Abbildung 2.1).

Kapitel 2 Einrichtung



Abbildung 2.1: Über den Eintrag Homebanking Kontakte (32-bit) in der Systemsteuerung bereiten Sie den Zugriff auf Ihre Kontodaten vor.

Mit einem Doppelklick auf diesen Eintrag öffnen Sie den Dialog aus Abbildung 2.2. Er enthält zunächst keine Einträge, was wir aber in den nächsten Schritten ändern werden – sofern Ihnen die notwendigen Daten bereits vorliegen. Das könnte zumindest der Fall sein, wenn Sie schon mit einer anderen Homebanking-Software arbeiten. Klicken Sie hier auf die Schaltfläche *Neu*, um einen neuen Kontakt anzulegen (der wiederum einem Benutzerkonto entspricht):

Homebanking-Kontakt erstellen

dministrator für Home	ebanking Ko	ntakte		X
Banking Ko	ntakte			
Folgende Henrikenskie				
Ausrufezeichen marki	) Kontakte sin erten Kontakte	d auf diese e müssen r	em System installier noch synchronisiert	t. Die mit einem werden.
Liste aller Kontakte	:			Aktualisieren
Kontakt	BL	Z E	Benutzerkennung	
Synchronisieren		Neu	Bearbeiten	Entfernen
Über Dete Desire Des	line (DDRAC		-	
Uber DataDesign Ban	IKING (DUDAC	<u></u>	Zur klassischen Ans	sicht wechsein

Abbildung 2.2: Ein Banking-Kontakt ist prinzipiell mit einem Konto bei einem Kreditinstitut identisch.

Der erste Schritt nach dem Willkommen-Dialog ist ganz einfach. Hier geben Sie einfach nur die Bankleitzahl Ihres Kreditinstituts ein (siehe Abbildung 2.3).

🗎 HBCI/FinTS-Kontakt
Einrichten eines neuen Homebanking Kontakts
Bitte geben Sie die achtstellige Bankleitzahl Ihres Kreditinstituts ein.
Bankleitzahl (BLZ):
35040038
Name des Kreditinstituts:
Commerzbank
Optional: Zugangsdaten manuell eingeben (für Experten) Wählen Sie diese Option nur, wenn Sie beim Einrichten des Kontakts die empfohlenen Einstellungen ändern möchten.
Optional: Proxy-Server verwenden Wählen Sie diese Option nur, wenn Ihre Internetverbindung einen sogenannten Proxy-Server verwendet an dem ggf. eine Anmeldung erforderlich ist.
Zurück Weiter Abbrechen

Abbildung 2.3: Als Erstes benötigt der Assistent die Bankleitzahl Ihres Kreditinstituts.

Kapitel 2 Einrichtung

Dann ist erstmal der Assistent gefragt, der die möglichen Zugangsarten für den Zugriff auf die Konten dieser Bank ermittelt (siehe Abbildung 2.4).

🗎 HBCI/FinTS-Kontakt	
Einrichten eines neuen Homebanking Kontakts	
Zugangsarten werden ermittelt. Dieser Vorgang kann einige Minuten dauern.	
Suche in BLZ Datenbank. HBCl Zugang wird geprüftPrüfung erfolgreich. Klicken Sie bitte auf weiter.	
Zurück Weiter Abbrechen	]

Abbildung 2.4: Anschließend prüft der Assistent die für das angegebene Kreditinstitut verfügbaren Zugangsarten.

Dabei gibt es verschiedene Varianten:

- » PIN/TAN: Die klassische Variante und von einigen Banken schon nicht mehr unterstützte Variante. Sie erfordert keine weiteren Hilfsmittel außer einer PIN-Nummer, mit der Sie sich bei der Bank anmelden, und eine Liste mit TANs (TransAktion-Nummer). Wenn Sie während einer Sitzung, für die Sie sich mit Ihrer PIN-Nummer angemeldet haben, Aktionen wie etwa eine Überweisung vornehmen möchten, brauchen Sie eine der auf der Liste enthaltenen TANs. Diese veraltete, aber wegen der geringen technischen Anforderungen auf Client-Seite immer noch sehr verbreitete Methode werden Sie vermutlich bereits einmal durchgeführt haben.
- » Chipkarte: Hierbei erhalten Sie von Ihrem Kreditinstitut einen Chipkartenleser und eine entsprechende Chipkarte, die für die Signatur verschlüsselter Nachrichten verwendet wird. Wichtig ist hier die Authentifizierung, also der Vorgang, bei dem sichergestellt wird, dass sich auch der entsprechende Kunde mit der Chipkarte anmeldet. Die Authentifizierung besteht hier schlicht darin, dass die Bank dem Kunden die Karte übergibt.
- » Ähnlich sieht es bei der Verwendung einer Schlüsseldatei aus: Auch diese hilft bei der Verschlüsselung der Nachrichten. Allerdings erfolgt die Authentifizierung hier dadurch, dass der Kunde den öffentlichen Part eines durch die notwendige Software erzeugten Schlüsselpaars

gleichzeitig auf elektronischem als auch auf postalischem Wege an die Bank schickt. Ein erfolgreicher Vergleich dieser beiden Schlüssel entspricht der Authentifizierung.

Wenn sichergestellt ist, dass nur der Benutzer die Chipkarte beziehungsweise den privaten Teil des Schlüsselpaares besitzt (dafür ist letztlich der Benutzer verantwortlich), kann dieser seine Nachrichten damit verschlüsseln und somit signieren.

Der Dialog aus Abbildung 2.5 stellt die möglichen Zugangsarten zur Auswahl.

🖼 HBCI/FinTS-Kontakt
Einrichten eines neuen Homebanking Kontakts
Das Kreditinstitut bietet mehrere Zugangsarten an. Bitte wählen Sie eine Zugangsart aus.
Zugang über
PIN/TAN (inkl. zwei-Schritt-TAN) Ihre Bank stellt ihnen für den Zugriff auf Ihr Konto einen Zugang mit PIN zur Verfügung. Zur Durchführung von Transaktionen benötigen Sie TANs (z. Bsp. auch iTAN, mobile TAN per SMS).
Chipkarte Für den Zugriff auf Ihr Konto verwenden Sie eine Chipkarte, mit der Aufträge elektronisch unterzeichnet werden.
Schlüsseldatei Für den Zugriff auf Ihr Konto verwenden Sie eine Schlüsseldatei, mit der Aufträge unterzeichnet werden.
<li>Zurück Weiter Abbrechen</li>

Abbildung 2.5: In diesem Fall ist der Zugang per PIN/TAN, Chipkarte und per Schlüsseldatei möglich.

Wir wählen zu Beispielzwecken den Eintrag *Schlüsseldatei* aus. Im folgenden Schritt geben Sie dann an, ob Sie bereits eine Schlüsseldatei besitzen oder nicht (siehe Abbildung 2.6).

Falls nicht, startet der oben beschriebene Vorgang – es wird ein Schlüsselpaar erzeugt und durch die Zusendung des öffentlichen Schlüssels auf zwei Wegen an die Bank wird dieser als Ihr Schlüssel authentifiziert.

Kapitel 2 Einrichtung



**Abbildung 2.6:** Möglicherweise verfügen Sie durch den Einsatz einer alternativen Homebanking-Software bereits über eine Schlüsseldatei – sonst müssen Sie eine neue anlegen.

Davon ausgehend, dass bereits eine Schlüsseldatei vorliegt, geben Sie den Speicherort dieser Datei an (siehe Abbildung 2.7). Stellen Sie sicher, dass das Medium mit dieser Datei (heutzutage wohl eher ein USB-Stick als eine Diskette) an einem geschützten Ort untergebracht ist.

Speichern Sie diese nicht auf der Festplatte!

Homebanking-Kontakt erstellen

HBCI/FinTS-Kontakt	<b>×</b>
Einrichten eines neuen Homebanking	g Kontakts
Wählen Sie bitte Ihre Schlüsseldatei aus.	
Dateiname:  c:\	Durchsuchen
Benutzername:	
Zurück Weiter	Abbrechen

Abbildung 2.7: Falls eine Schlüsseldatei vorhanden ist, geben Sie ihren Speicherort hier an.

Zum Anlegen des Kontakts fehlt nun noch die Angabe der Passphrase, die Sie beim Erstellen des Schlüsselpaars angegeben haben. Diese geben Sie im Dialog aus Abbildung 2.8 ein.

Kapitel 2 Einrichtung



Abbildung 2.8: Wenn Sie bereits eine Schlüsseldatei besitzen, brauchen Sie nur noch das Passwort einzugeben.

Es fehlen noch ein paar Kundendaten, die der Dialog aus Abbildung 2.9 abfragt.

Gleich im Anschluss nimmt der Assistent nochmals Kontakt zum Server des Kreditinstituts auf und ruft die letzten erforderlichen Informationen ab (siehe Abbildung 2.10).

Homebanking-Kontakt erstellen

🖼 HBCI/FinTS-Kontakt
Einrichten eines neuen Homebanking Kontakts
Bitte geben Sie ihre Kundendaten für den Zugang bei der "Commerzbank" ein.
Kontonummer:
100000000000000000000000000000000000000
Kunden-ID:
10000101000000
Kontaktname:
Commerzbank
Später synchronisieren
<li><zurück abbrechen<="" td="" weiter=""></zurück></li>

Abbildung 2.9: Jetzt nur noch schnell die Kundendaten eingeben, die Sie ebenfalls von Ihrer Bank erhalten ...

BBCI/FinTS-Kontakt	×
Einrichten eines neuen Homebanking Kontak	ts
Die Zugangsdaten werden synchronisiert. Dieser Vorgang kann einige Minuten dauern.	
Kontakt wird synchronisiert. Bitte warten	*
	Ŧ
Zurück Weiter Abbrecher	

Abbildung 2.10: ... und synchronisieren – fertig!

Mit diesen im Gepäck erstellt er schließlich den neuen Eintrag in der Liste der Bank-Kontakte. Wenn Sie diesen Kontakt markieren, können Sie ihn neu synchronisieren (was von Zeit zu Zeit nötig sein kann, aber auch per Code durchführbar ist), ihn bearbeiten oder auch entfernen (sieKapitel 2 Einrichtung

he Abbildung 2.11). Natürlich können Sie hier auch noch weitere Kontakte anlegen – vielleicht haben Sie ja etwa sowohl ein privates als auch ein Geschäftskonto.

lministrator für Homeban	king Kontakte		<b>×</b>
Banking Konta	akte		
Folgende Homebanking Kon Ausrufezeichen markierten	takte sind auf di Kontakte müsse	esem System installie n noch synchronisier	rt. Die mit einem t werden.
Liste aller Kontakte:			Aktualisieren
Commerzbank	35040038	Denutzerkennung	
Synchronisieren	Neu	Bearbeiten	Entfernen
Über DataDesign Banking	(DDBAC)	Zur klassischen An	sicht wechseln
2			Schließen

Abbildung 2.11: Anschließend finden Sie im *Administrator für Homebanking Kontakte* einen ersten Eintrag, dem Sie bei Bedarf noch weitere hinzufügen können.

### 2.2 Konfiguration bearbeiten

Nachdem Sie ein oder mehrere Kontakte angelegt haben, können Sie diese in der Übersicht aus Abbildung 2.12 anzeigen.

Konfiguration bearbeiten

omebanking	g Kontakte Herstellerhinweis		
Kontakte	Folgende Homebanking Kontak einem Ausrufungszeichen marki werden.	te sind auf dies erten Kontakte	em System installiert. Die mit müssen noch synchronisiert
Kontakt		BLZ	Benutzerkennung
🚽 Co	mmerzbank	35040038	
🔛 Da	taDesign Demobank FinTS3	70000997	11110
	audin, pirva		
	Neu	Entfe	emen) Bearbeiten
Aktionen	Folgende Aktionen können Sie r Kontakt durchführen:	Entfe	emen) Bearbeiten ewählten Homebanking
Aktionen	Folgende Aktionen können Sie r Kontakt durchführen:	mit dem oben g	emen ) Bearbeiten ewählten Homebanking PIN (Passwort) ändem

Abbildung 2.12: Übersicht aller verfügbaren Homebanking-Kontakte

#### 2.2.1 Kontakt-Informationen

Nach der Auswahl eines der Einträge können Sie diesen mit den unter der Liste befindlichen Schaltflächen entfernen oder bearbeiten. Wenn Sie auf *Bearbeiten* klicken, erscheint ein weiterer Dialog, der alle notwendigen Informationen auf mehreren Registerseiten bereitstellt. Für das Beispielkonto der Firma *DataDesign* sieht dieser Dialog beispielsweise wie in Abbildung 2.13 aus. Kapitel 2 Einrichtung

Eigenschaft	en von Homebanking Kontakt	×
Kontakt	Konten Verbindung Aktionen	TAN - Verfahren
Zugangs	daten	
	Homebanking Kontakt:	DataDesign Demobank FinTS3
	Protokoll:	FinTS 3.0
	Bankleitzahl:	70000997
	Benutzerkennung:	0.00
	Kunden-ID:	
	Sicherheitsmedium:	HBCI+ mit PIN/TAN
		$\searrow$
		OK Abbrechen Hilfe

Abbildung 2.13: Allgemeine Informationen zum Kontakt

Auf der ersten Seite namens *Kontakt* finden Sie die allgemeinen Informationen wie die Bezeichnung des Kontakts, die Bankleitzahl, die Kontonummer und das Sicherheitsmedium. Außerdem stellen Sie hier das Protokoll ein, mit dem Sie mit der Bank kommunizieren. Um dieses Protokoll zu ändern, klicken Sie auf die Schaltfläche mit den drei Punkten rechts neben der Eigenschaft Protokoll. Dies öffnet den Dialog aus Abbildung 2.14.

Konfiguration bearbeiten

HBCI Versi	on 💦 💌
HBCI Vers	ACHTUNG: Die Einstellung der korrekten HBCI Version ist Voraussetzung für eine erfolgreiche Auftragsabwicklung mit dem HBCI Banksystem/ Wählen Sie die von Ihrem Kreditinstitut unterstützte HBCI Version. Falls Sie sich nicht sicher sind, wählen Sie die HBCI Version 2.2.
	Standardversion, für allgemeinen Zahlungsverkehr. HBCI 2.1 Mit zusätzlicher Unterstützung des Wertpapier-Geschäfts. HBCI 2.2
	Mit erweiterter Unterstützung des Wertpapier-Geschäfts. Von den meisten Kreditinstituten angebotene Version. FinTS 3.0 Nur für PIN/TAN und Signaturkarten nach FinTS 3.0 erforderlich. Wählen Sie diese Version nur, wenn Sie von Ihrem Kreditinstitut explizit verlangt wird.
	FinTS 4.0 Nur erforderlich falls dies von Ihrem Kreditinstitut ausdrücklich verlangt wird.

Abbildung 2.14: Auswahl des Protokolls für die Kommunikation mit der Bank

Die zweite Registerseite des Dialogs liefert eine Liste aller Konten dieses Kontakts (siehe Abbildung 2.15). Ein Doppelklick auf eines der Konten oder das Markieren eines Kontos und anschließendes Betätigen der Schaltfläche *Bearbeiten…* zeigen Sie die Details zum jeweiligen Konto an (siehe Abbildung 2.16). Mit der Schaltfläche *Entfernen* löschen Sie das aktuell markierte Konto.

Kapitel 2 Einrichtung

Eigenschaften Kontakt Koni	von Homeban ten Verbindun	king Kontakt g   Aktionen   TAN - V	erfahren	<b>X</b>
	Konto	Kontoname	Kundenname	Whg Kunden-
		Girokonto Girokonto (verzögert) Depot	USER_ USER_ USER_	EUR EUR EUR
	•	11	1	4
		Neu	Entfemen	Bearbeiten
			OK Abbrech	nen Hilfe

Abbildung 2.15: Übersicht der vorhandenen Konten

Kontodate	'n	
Konto –	Bankleitzahl: Kontonummer: Kunden-ID: Kontobezeichnung: Kontowährung: Kundenname:	70000997
		OK Abbrechen Hilfe

Abbildung 2.16: Eigenschaften eines Kontos

Die Registerseite *Verbindung* aus Abbildung 2.17 liefert die Informationen über den Bank-Server – zum Beispiel die Internetadresse für den Zugriff oder den entsprechenden Port.

Konfiguration bearbeiten

Eigenschafter	n von Homebanking Ko	ntakt	×
Kontakt Ko	nten Verbindung Aktio	nen TAN - Verfahren	
Internetvert	pindung		- 1
	Internet Adresse:	https://fints.datadesign.de/j2hbci-gateway/j2hbci	
	Port:		
	Base64-Encoding:	Aktiviert	
	Socks5 Proxy		- II
	Falls Sie sich hinter eir verwendet, dann müss gegebenfalls Benutzer	nem Proxyserver befinden der das Socks5 Protokoll sen sie hier den Namen des Proxyservers angeben und mame und Passwort. venden	
	Proxyserver:		וור
	Port:		5
	Benutzemame:		
	Passwort:		
		OK Abbrechen Hilf	e

Abbildung 2.17: Informationen über die Verbindung zum Bank-Server

#### 2.2.2 PIN- und TAN-Funktionen

Interessant wird es auf der Registerseite *Aktionen* aus Abbildung 2.18. Hier finden Sie eine ganze Reihe von Schaltflächen, mit denen Sie die folgenden Aktionen durchführen können:

- » *PIN ändern...*: Ein Klick auf diese Schaltfläche öffnet den Dialog aus Abbildung 2.19. Hier können Sie den PIN-Code für das Onlinebanking ändern. Dies sollte man beispielsweise tun, wenn der PIN-Code möglicherweise anderen Personen bekannt geworden ist.
- » PIN sperren: Hiermit sperren Sie den kompletten Onlinezugang.
- » *PIN-Sperre aufheben*: Diese Schaltfläche entsperrt den Onlinezugang wieder. Hierzu benötigen Sie die PIN und eine gültige TAN.
- » TAN-Liste anfordern: Hiermit können Sie sich eine neue TAN-Liste zusenden lassen.
- » TAN-Liste aktivieren: Mit diesem Befehl aktivieren Sie die neue TAN-Liste.
- » TAN-Liste sperren: Diese Schaltfläche sperrt eine TAN-Liste.
- » *TAN-Information*: Dies ist eine mächtige Funktion. Sie können damit auf Informationen rund um und auf den Inhalt der TAN-Listen zugreifen mehr dazu weiter unten.

Kapitel 2 Einrichtung

Eigenschaften von Homebanking Kontakt
Kontakt Konten Verbindung Aktionen TAN - Verfahren
PIN-Verwaltung Andem Sie Ihre PIN bei Erstzugang oder gegebenenfalls bei Verlust bzw. Verdacht auf Kenntnisnahme durch Unbefugte.
PIN ändem PIN sperren PIN-Sperre aufheben TAN-Verwaltung
Fordem Sie hier eine neue TAN-Liste von Ihrem TAN-Liste anfordem Kreditinstitut an.
Um eine TAN-Liste zu aktivieren, benötigen Sie je eine TAN der alten und der neuen zu aktivierenden TAN-Liste. TAN-Liste aktivieren
Sperren Sie hier ihre aktive TAN-Liste oder geben Sie die TAN-Liste sperren Nummer der zu sperrenden TAN-Liste an (optional).
Fordem Sie hier Informationen über Ihre verbrauchten TAN-Informationen TANs an.
OK Abbrechen Hilfe

Abbildung 2.18: Die Registerseite Aktionen hält einige weitere Möglichkeiten bereit.

Online-PIN ä	indern	<b>x</b>
Ge Fe bis	eben Sie bitte Ihre neue gewünschte Online-PIN (beide alder) ein und klicken Sie auf (DK). Zum Beibehalten Ihrer sherigen Online-PIN klicken Sie bitte auf (Abbrechen).	OK Abbrechen
Kontakt:	DataDesign Demobank FinTS3	Hilfe
Alte Online-P	'IN: [	]
Neue Online-	-PIN:	]
Wiederholun	g:	

Abbildung 2.19: Ändern des Online-PINs

#### 2.2.3 TAN-Informationen abfragen

Mit der Schaltfläche *TAN-Informationen* des Dialogs *Eigenschaften von Homebanking Kontakt* starten Sie den Abruf eines HTML-Dokument mit einer TAN-Liste und weiteren Eigenschaften. Wir wollen bei dieser Gelegenheit einmal beispielhaft zeigen, wie das Zwei-Schritt-TAN-Verfahren funktioniert. In diesem Fall erscheint zunächst ein Dialog, der den PIN-Code abfragt (siehe Abbildung 2.20). Nach der Eingabe nimmt die Komponente erstmalig Kontakt zum Bank-Server auf.

Konfiguration bearbeiten

HBCI/FinTS-Kontakt	×
PIN (Passwort) Eingabe	
Für die gewünschte Funktion wird eine gültige F Homebanking Kontakt: DataDesign Demobank FinTS3 TAN Verfahren: 901 mobile TAN PIN (Passwort):	IN (Passwort) benötigt. Verfügbare TAN Medien: TAN Liste: 1138157702 ▼ 1 2 3 4 5 6 7 8 9 C 0 ←
Zurück Weiter	Abbrechen

Abbildung 2.20: Eingabe der PIN im PIN/TAN-Verfahren, um die TAN-Informationen abzurufen

Dies zieht nach sich, dass ein weiterer Dialog erscheint und - in diesem Fall - den per SMS verschickten TAN abfragt (siehe Abbildung 2.21).

wei-Schritt-TA	AN-Verfahren	×
TAN - E	ingabe	
Kontakt:	DataDesign Demobank FinTS3	3
Verfahren:	mobile TAN	6
TAN:	78	9
Bitte beacht SMS Die TAN wu	ten Sie folgende Hinweise:	<u>۲</u>
		Ŧ
2	Abbrechen OK	

Abbildung 2.21: Eingabe der TAN im PIN/TAN-Verfahren

Kapitel 2 Einrichtung

Nach der Eingabe des TAN erfolgt eine weitere Verbindung mit dem Bank-Server. Nach erfolgreichem Verbindungsaufbau bietet der entsprechende Dialog lediglich die Schaltfläche *Schließen* an (siehe Abbildung 2.22).

			<b>.</b>		
Homebanking Aktion					
m	Kontakt:	DataDesign Demobank FinTS3			
2.99	Aktion:	TAN Verbrauchsinformationen anfordem			
	ngsprotokoll:				
	<ul> <li>Die Verbindung zum Homebanking System des Kreditinstituts wird aufgebaut</li> <li>TAN-Verbrauchsinformationen anzeigen wird an Banksystem übertragen.</li> <li>Die Verbindung zum Homebanking System des Kreditinstituts wird abgebaut</li> <li>Die HBCI Online Aktion wurde beendet. Klicken Sie jetzt auf Schließen.</li> </ul>				
© DataDe	sign AG	Schließen			

Abbildung 2.22: Zweite Kontaktaufnahme der DDBAC-Komponente mit dem Bank-Server

In der Zwischenzeit hat die DDBAC-Komponente ein HTML-Dokument heruntergeladen und in einem temporären Verzeichnis gespeichert. Dieses wird im Browser angezeigt (siehe Abbildung 2.23).

Neuer Tab × C SEPA%20Segr × C DataDesign DI × C TAN-Verbrau: × TAN-Verbrau: ×								
C ile:///C:/Users/Andre/AppData/Local/Temp/TANList.html			Ś					
TAN-Verl	TAN-Verbrauchsinformation							
Das folgende Dokument enthält eine Übersicht über Ihre verbrauchten TANs.								
Homebank	ing-Kontakt: DataDesign Demobank FinTS3							
Kreditinsti	ut (BLZ): 70000997							
Benutzerk	ennung:							
TAN-Liste Nummer 6 (Aktiv) vom (unbekannt)								
TAN	verweildung	Datum	Unrzeit					
123456	Zahlungsverkehrstransaktion (Uberweisung/Lastschrift)							
123456	Wertpapiertransaktion (Neuanlage/Änderung/Löschung)							
Diese TAN-Verbrauchsinformation wurde von den DataDesign Banking Application Components (DDBAC) erstellt.								


Speicherort der Konfiguration

### 2.2.4 TAN-Verfahren auswählen

Auf der letzten Registerseite des Dialogs *Eigenschaften von Homebanking Kontakte* können Sie die zur Verfügung stehenden TAN-Verfahren einsehen und eines davon als aktuelles Verfahren auswählen (siehe Abbildung 2.24).

Eigenschaften von Homebanking Kontakt
Kontakt Konten Verbindung Aktionen TAN - Verfahren
TAN - Verfahren Ihr Kreditinstitut stellt folgende Zwei-Schritt-TAN-Verfahren zur Verfügung. Sie können hier ihr hevorzurdes TAN Verfahren festlenen
© iTAN
mobile TAN
⊚ mobile TAN
Smart TAN optic
⊘ klassisches PIN/TAN Verfahren
Synchronisieren Sie ihren Kontakt um die Daten zu aktualisieren.
OK Abbrechen Hilfe

Abbildung 2.24: Auswahl des zu verwendenden TAN-Verfahrens

# 2.3 Speicherort der Konfiguration

Die über die Systemsteuerung eingegebenen Daten landen in einer Textdatei, die in einem speziellen Verzeichnis gespeichert wird. Unter Windows 7 finden Sie diese Textdatei namens *ddusers. dat* beispielsweise im Verzeichnis *C:\Users\<Benutzername>\AppData\Roaming\DataDesign\ DDBAC*. Es kann sein, dass diese Datei in einem anderen Verzeichnis gespeichert wird. In diesem Fall finden Sie den Pfad in der Registry unter dem Eintrag *HKEY\_CURRENT\_USER\Software\ DataDesign\DDBAC* oder *HKEY\_LOCAL\_MACHINE\Software\DataDesign\DDBAC*, und zwar unter dem Namen *DataDir*. Standardmäßig ist dieser Bereich der Registry leer. Die Textdatei sieht beispielsweise wie in Abbildung 2.25 aus.

#### Kapitel 2 Einrichtung



Abbildung 2.25: Die Benutzerdaten, die über die Systemsteuerung angelegt wurden, in einer Textdatei

Wenn Sie die Daten also auf einem Rechner bereits angelegt haben und diese etwa bei einem Umzug auf einen anderen Rechner mitnehmen möchten, kopieren Sie diese Datei in das entsprechende Verzeichnis auf dem neuen Rechner. Danach müssen Sie alle Kontakte allerdings nochmals synchronisieren.

# 2.4 Debugging/Protokolldatei

Die DDBAC-Komponenten bieten die Möglichkeit, Informationen zu allen durchgeführten Vorgängen in einer Protokolldatei zu erfassen. Um diese Option zu aktivieren, wechseln Sie etwa in der klassischen Ansicht des *Administrators für Homebanking Kontakte* zur Registerseite *Herstellerhinweise*. Hier aktivieren Sie die Option *HBCI Protokolldatei aufzeichnen* (siehe Abbildung 2.26). Die Protokolldatei enthält nun

# 3 Grundlagen der DDBAC

Mit der *DDBAC* (*DataDesign Banking Application Component*) bietet die *B+S Banksysteme AG* eine Sammlung von Software-Komponenten an, mit der Sie Onlinebanking-Funktionen etwa in eigenen Access-Anwendungen implementieren können. Wie üblich lassen sich die in dem Paket enthaltenen Funktionen wahrscheinlich auch selbst per VBA programmieren, aber wer will erstens diesen Aufwand betreiben und sich zweitens auch noch immer darum kümmern, dass die Funktionen den aktuellen Gegebenheiten entsprechen? Es gibt Anwendungsfälle im Leben eines Access-Programmierers, die man einfach den jeweiligen Spezialisten überlassen sollte. Vor allen, wenn es – wie es bei Drucklegung dieses Buchs der Fall ist – unter *http://www.amvshop. de* recht günstige Lizenzen für diese Komponente gibt.

Die *DDBAC*-Komponenten basiert auf dem deutschen Standard *FinTS* (*Financial Transaction Services*, früher *HBCI*, *Homebanking Computer Interface*). Mit den *DDBAC*-Komponenten lässt sich eine Vielzahl von Vorgängen programmieren – zum Beispiel die folgenden:

- » Ermittlung von Kontoständen (siehe »Kontostand ermitteln«, Seite 173)
- » Abruf von Kontoumsätzen (siehe »Umsätze einlesen«, Seite 207)
- » Durchführung von SEPA-Überweisungen (siehe »SEPA-Überweisung«, Seite 251)
- » Durchführen von Terminüberweisungen (siehe »SEPA-Terminüberweisungen«, Seite 281)
- » Anlegen von Daueraufträgen (siehe »SEPA-Daueraufträge«, Seite 307)
- » Durchführen von SEPA-Lastschriften (siehe »Lastschriften mit SEPA«, Seite 353)
- » Durchführen von SEPA-Sammellastschriften (siehe »SEPA-Lastschriften verwalten« ab Seite 385)

Die folgenden Standards werden durch die DDBAC realisiert:

- » HBCI 2.0
- » HBCI 2.0.1
- » HBCI 2.1
- » HBCI 2.2
- » HBCI+ (PIN/TAN-erweitert)
- » FinTS 3.0
- » FinTS 4.0

Die DDBAC will mit jedem HBCI/FinTS-Banksystem kompatibel sein und wird vom Hersteller ständig weiterentwickelt, um dies zu gewährleisten.

# 3.1 VBA-Zugriff

Damit Sie einfach per VBA auf die *DDBAC*-Komponenten zugreifen können, liegen diese in Form von Bibliotheken vor, die Sie ganz einfach über den *Verweise*-Dialog in Ihre Anwendung einbinden können. Dazu müssen Sie die *DDBAC*-Komponenten natürlich zuvor installiert haben. Abbildung 3.1 zeigt die verschiedenen Bibliotheken im *Verweise*-Dialog.

Verfügbare Verweise:	OK Abbrechen
DataDesign DDBAC CT-API Card Terminal Component DataDesign DDBAC DDV Chipcard Security Componen DataDesign DDBAC DTAUS Format Component DataDesign DDBAC FinTS 4.0	Durchsuchen
DataDesign DDBAC HBCI Banking Application Compor DataDesign DDBAC HBCI Service Provider Component DataDesign DDBAC HTTP Transport Component DataDesign DDBAC PC/SC Smart Card API Componer DataDesign DDBAC RDH Security Componer DataDesign DDBAC SAC Chipcard Security Component DataDesign DDBAC SNU.I.F.T. Format Component DataDesign DDBAC SW.I.F.T. Format Component V III V Component	Hilfe
Microsoft Office 14.0 Access database engine Object Library	SOFT SHARED\C
Sprache: Voreinstellung	

Abbildung 3.1: Einbinden der DDBAC-Bibliotheken

Diese Bibliotheken enthalten alle Objekte, Eigenschaften und Methoden, um Onlinebanking-Anwendungen mit VBA zu programmieren. Für eine bessere Übersicht haben wir die Objektmodelle der einzelnen Bibliotheken in HTML-Dateien exportiert. Sie finden diese im Download zum Buch unter dem Verzeichnis *Typelibs*. Abbildung 3.2 zeigt beispielsweise die Elemente der Bibliothek *DataDesign DDBAC HBCI Banking Application Components*. Die Objektmodelle der übrigen Bibliotheken finden Sie in weiteren Dateien.

#### VBA-Zugriff

BankingApplicationComp ×
$\leftrightarrow \Rightarrow \mathbf{C}$ $[]$ file:///C:/Daten/Buecher/OnlinebankingMitAccess/DataDesignDDBACHBCIBankingApplicationComponents.htm $c_{2}$ ]
☑       Diese Seite ist auf Englisch → Soll sie übersetzt werden?       Übersetzen       Nein       Optionen → ×
This file was automatically generated by <b>Oberon TLB Tools</b> Type library name: <b>BaukingApplicationComponents</b> Description: <b>DataDesign DDBAC HBCI Banking Application Components</b> Version: 1.0
Oberon TLB Tools © Copyright 2001 by <u>Alex Vakulenko</u>
Index
BACAccountBACMessageBACSyntaxIBACCardTerminalManagerBACAccountDataBACMessageBufferBACSyntaxElementIBACSecurityBACBankDataBACSegmentBACSyntaxSegmentIBACSecurityBACBankingBACSegmentBACTransactionIBACSecurityBACContactBACSepaMessageBACTransactionIBACSegmentBACContactsBACSepaOrderBIZInfoIBACSegmentBACCustomerBACSepaOrderIBACCardTerminalBACCustomerBACSignaturesIBACCardTerminalBACCustomersBACSignaturesIBACCardTerminalBACDataObjectBACSignaturesIBACCardTerminalBACDialogBACStorageIBACCardTerminal
Class BACAccount         Property AccountNumber As String [r/w]         Property AccountNumber As String [r/w]         Property AcctName As String [r/w]         Property AcctName As String [r/w]         Property AvailableTransactions As Object [r/o]         Property BankCode As String [r/w]         Property BankCode As String [r/w]         Property BIC As String [r/w]         Property BIC As String [r/w]

Abbildung 3.2: HTML-Datei mit allen Objekten, Methoden und Eigenschaften einer der DDBAC-Bibliotheken

In den Beispielen zu diesem Buch verwenden wir die folgenden Bibliotheken:

- » DataDesign DDBAC HBCI Banking Application Components: Enthält fast alle Objekte, die für übliche Vorgänge nötig sind
- » DataDesign DDBAC S.W.I.F.T. Format Component: Liefert Funktionen zum Parsen von Kontoumsätzen
- » DataDesign DDBAC+ PIN/TAN Security Component: Enthält Elemente, die zur Programmieren einer individuellen Eingabe von PIN/TAN nötig sind.
- » DataDesign DDBAC IBAN Converter: DLL, die das Konvertieren von BLZ und Kontonummern in IBAN und BIC erlaubt und mehr.

#### Konten einrichten

Wie Sie weiter oben bereits erfahren haben, brauchen Sie sich um die Einrichtung der Benutzer und Konten gar nicht zu kümmern – eine installierte DDBAC liefert über die Systemsteuerung bereits alle benötigten Elemente.

Wie dies gelingt, haben Sie bereits im Kapitel »Einrichtung«, Seite 17 erfahren.

# 3.2 Objektmodell der DDBAC

Die DDBAC-Komponente *DataDesign DDBAC HBCI Banking Application Components* liefert die meisten der in diesem Buch beschriebenen Objekte. Die folgenden Abschnitte stellen die einzelnen Objekte mit ihren Auflistungen, Methoden und Eigenschaften vor.

Im Folgenden schauen wir uns einige Objekte an, die für die nachfolgenden Beispiele wichtig sind. Zunächst erhalten Sie eine kurze Beschreibung zu den einzelnen Objekten:

- » BACBanking: Dies ist das Hauptobjekt beim Arbeiten mit der DDBAC-Bibliothek. Sie können damit auf die Onlinebanking-Kontakte zugreifen (BACCustomer) und diese anlegen oder löschen, den über die Systemsteuerung verfügbaren Dialog zum Verwalten von Kontakten öffnen oder Optionen einstellen.
- » BACCustomer: Das BACCustomer-Objekt repräsentiert einen Kontakt, also eine Kunde-Bank-Beziehung. Die Identifizierung eines solchen Kontaktes erfolgt über die Länderkennung (280 für Deutschland), die Bankleitzahl und die UserID. Letztere kann mit einer Kontonummer identisch sein, allerdings kann es zu jedem Kontakt auch mehrere verschiedene Konten geben – die wiederum verschiedenen Typs sein können.
- » BACDialog: Dieses Objekt stellt die Verbindung zwischen Client und Server her, wobei Ihre Anwendung dem Client und der Bankserver dem Server entspricht. Mit dem BACDialog-Objekt können Sie den Dialog starten und beenden und Nachrichten oder Segmente mit dem Bankserver austauschen.
- » *BACMessage*: Dieses Objekt liefert die Antwort auf die Ausführung eines Geschäftsvorgangs und enthält weitere Objekte mit den benötigten Informationen.
- » BACTransaction: Das BACTransaction-Objekt liefert die eigentlichen Transaktions-Informationen, also beispielsweise den aktuellen Kontostand oder die Konto-Umsätze.
- » BACSegment: Das BACSegment-Objekt ist ein Objekt, das einem HBCI-Datensatz entspricht. Sie können diesem die Parameter zuweisen, die das Segment beinhalten soll.

**Das BACBanking-Objekt** 

# 3.3 Das BACBanking-Objekt

Das Objekt *BACBanking* ist das Hauptobjekt der Bibliothek *DataDesign DDBAC HBCI Banking Application Components*. Es bietet die folgenden Methoden und Eigenschaften, die wir uns im Anschluss im Detail und mit Beispielen ansehen:

- » Property Customers As BACCustomers [r/o]
- » Property Options(ByVal sOptionName As String) As String [r/w]
- » Sub DeleteCustomer(ByVal ICountryCode As Long, ByVal sBankCode As String, ByVal sUserID As String)
- » Function GetSmartcardSecurityProgID(ByVal piCT As IBACCardTerminal) As String
- » Function NewCardTerminal() As IBACCardTerminal
- » Function NewCustomer(ByVal ICountryCode As Long, ByVal sBankCode As String, [ByVal sUserID As String = ""]) As BACCustomer
- » Function NewDialog(ByVal piCustomer As BACCustomer, ByVal piTransport As IBACTransport, [ByVal piSecurity As IBACSecurity = 0]) As BACDialog
- » Function NewSegment(ByVal sSegmentType As String, [ByVal IVersion As Long = 0]) As IBACSegment
- » Sub RunContactsCPL()
- » Function RunNewContactWizard() As BACCustomer
- » Function SynchronizeCustomer(ByVal piCustomer As BACCustomer) As Boolean

### 3.3.1 Einfacher Zugriff auf das BACBanking-Objekt

Da wir in den folgenden Beispielen und bei Verwendung der *DDBAC*-Bibliothek insgesamt sehr oft auf eine Instanz des *BACBanking*-Objekts zugreifen werden, legen wir dazu eine entsprechende Funktion an. Diese speichert beim ersten Zugriff einen Verweis auf das *BACBanking*-Objekt in der wie folgt im Standardmodul *mdlOnlinebanking\_Objekte* deklarierten Variablen:

Private m Banking As BACBanking

Die Funktion heißt schlicht und einfach GetBanking und sieht wie folgt aus:

```
Public Function GetBanking() As BACBanking
    If m_Banking Is Nothing Then
        Set m_Banking = New BACBanking
    End If
    Set GetBanking = m_Banking
```

End Function

Die Funktion prüft, ob *m\_Banking* schon gefüllt ist. Falls nicht, erstellt die Funktion ein neues Objekt des Typs *BACBanking* und referenziert es mit der Variablen *m\_Banking*. Die Funktion liefert mit *GetBanking* dann den in *m\_Banking* enthaltenen Verweis.

Diese Funktion liefert beispielsweise den Vorteil, dass Sie nicht jedes Mal den Code zum Instanzieren eines *BACBanking*-Objekts benötigen. Nehmen wir an, Sie benötigen Code, um den Kontakte-Manager per VBA zu öffnen. Dann würden Sie ohne diese Funktion die folgende Prozedur nutzen:

```
Public Sub KontakteVerwalten()
   Dim objBanking As BACBanking
   Set objBanking = New BACBanking
   With objBanking
        .RunContactsCPL
   End With
End Sub
```

Wenn Sie aber zuvor die Funktion *GetBanking* eingerichtet haben, kommen Sie mit diesen Codezeilen aus:

```
Public Sub KontakteVerwaltenKurz()
GetBanking.RunContactsCPL
End Sub
```

Praktischerweise erhalten Sie über die Funktion GetBanking auch Zugriff auf IntelliSense:

Direktbereich			X
GetBanki	ng. Customers		<u> </u>
	GetSmartcardSecurityProgID     NewCardTerminal		
	<ul> <li>NewCustomer</li> <li>NewDialog</li> </ul>		_
•	Segment	<b>T</b>	<u> </u>

Abbildung 3.3: IntelliSense bei Verwendung der GetBanking-Funktion

### 3.3.2 Kontaktverwaltung öffnen

Die Methode des *BACBanking*-Objekts zum Aufruf des Dialogs zum Verwalten von *Customer*-Objekten kennen Sie ja somit bereits. *RunContactsCPL* öffnet den Dialog aus Abbildung 3.4.

**Das BACBanking-Objekt** 

olgende Homebanking Kontal	kte sind auf di	esem System installie	rt. Die mit einem
lusrufezeichen markierten Ko .iste aller Kontakte:	ontakte müsse	en noch synchronisier	t werden.
Kontakt	BLZ	Benutzerkennung	
Commerzbank			
DataDesign Demoban.		111100	
DataDesign Demoban.			
DataDesign Demoban.		101100	
			<b>E</b> 12

Abbildung 3.4: Dialog zum Verwalten der Banking-Kontakte

### 3.3.3 Neuen Kontakt einrichten

Die Funktion *RunNewContactWizard* startet den Dialog aus Abbildung 3.5, den Sie sonst nur über die Schaltfläche *Neu* des Dialogs *Banking Kontakte* anzeigen können. Der benötige Code sieht, mit unserer vereinfachenden Funktion *GetBanking*, wie folgt aus:

```
GetBanking.RunNewContactWizard
```

Kapitel 3 Grundlagen der DDBAC

🔁 HBCI/FinTS-Kontakt
Einrichten eines neuen Homebanking Kontakts
Bitte geben Sie die achtstellige Bankleitzahl Ihres Kreditinstituts ein.
Bankleitzahl (BLZ) oder BIC:
Name des Kreditinstituts:
<ul> <li>Optional: Zugangsdaten manuell eingeben (für Experten)</li> <li>Wählen Sie diese Option nur, wenn Sie beim Einrichten des Kontakts die empfohlenen Einstellungen ändern möchten.</li> <li>Optional: Proxy-Server verwenden</li> </ul>
Wählen Sie diese Option nur, wenn Ihre Internetverbindung einen sogenannten Proxy-Server verwendet an dem ggf. eine Anmeldung erforderlich ist.
Zurück Weiter Abbrechen

Abbildung 3.5: Dialog zum Einrichten eines neuen Homebanking-Kontakts

Bei *RunNewContactWizard* handelt es sich um eine Funktion, die den obigen Dialog aufruft und nach erfolgtem Anlegen des neuen Kontakts einen Verweis auf das neue *BACContact*-Objekt zurückliefert.

Auf dieses können Sie dann beispielsweise wie folgt zurückgreifen, um etwa die Bankleitzahl mit der Eigenschaft *BankCode* zu ermitteln:

```
Public Sub KontaktAnlegen()
   Dim objContact As BACContact
   Set objContact = GetBanking.RunNewContactWizard
   Debug.Print objContact.BankCode
End Sub
```

Das BACContact-Objekt schauen wir uns weiter unten im Detail an.

### 3.3.4 Die Customers-Auflistung

Mit der *Customers*-Auflistung erhalten Sie über das *BACBanking*-Objekt Zugriff auf die *BACCus-tomer*-Objekte. Am einfachsten geben Sie zunächst einmal die Anzahl der enthaltenen *BACCus-tomer*-Objekte aus – diesmal noch mit Deklaration und Instanzierung des *BACBanking*-Objekts:

```
Public Sub AnzahlCustomers()
  Dim objBanking As BACBanking
  Set objBanking = New BACBanking
```

**Das BACBanking-Objekt** 

```
With objBanking
Debug.Print "Anzahl Customers: " & .Customers.count
End With
End Sub
```

#### Einfacher geht es so:

```
Public Sub AnzahlCustomersKurz()
    Debug.Print GetBanking.Customers.count
End Sub
```

Weitere Aktionen zur *Customers*-Auflistung finden Sie weiter unten unter »Das BACCustomer-Objekt«, Seite 61.

#### **BACCustomer-Objekt referenzieren**

Ein spezielles *BACCustomer*-Objekt können Sie über die *Item*-Eigenschaft referenzieren. Wenn Sie etwa auf Eigenschaften wie die Bankleitzahl des *BACCustomer*-Objekts mit dem Index *0* zugreifen wollen, erreichen Sie dies so:

```
Public Sub BestimmterCustomer()
   Dim objCustomer As BACCustomer
   Set objCustomer = GetBanking.Customers.Item(0)
   Debug.Print objCustomer.BankCode
End Sub
```

#### **BACCustomer-Objekte durchlaufen**

Die einzelnen Elemente der *Customers*-Auflistung können Sie auf verschiedene Arten durchlaufen. Am einfachsten gelingt dies über die *For Each*-Schleife wie im folgenden Beispiel:

```
Public Sub CustomerDurchlaufenForEach()
   Dim objCustomer As BACCustomer
   For Each objCustomer In GetBanking.Customers
        Debug.Print objCustomer.BankCode
   Next objCustomer
End Sub
```

Dies gelingt auch über den Index, den Sie über eine For...Next-Schleife wie folgt durchlaufen:

```
Public Sub CustomerDurchlaufenForNext()
  Dim objCustomer As BACCustomer
  Dim i As Integer
  For i = 0 To GetBanking.Customers.count - 1
    Set objCustomer = GetBanking.Customers.Item(i)
    Debug.Print objCustomer.BankCode
```

Next i End Sub

Die *Customers*-Auflistung können Sie auch zuvor mit einer eigenen Variablen des Datentyps *BACCustomer* referenzieren:

```
Public Sub CustomerDurchlaufenForNextII()
   Dim objCustomer As BACCustomer
   Dim objCustomers As BACCustomers
   Dim i As Integer
   Set objCustomers = GetBanking.Customers
   For i = 0 To objCustomers.count - 1
      Set objCustomer = objCustomers.Item(i)
      Debug.Print objCustomer.BankCode
   Next i
End Sub
```

### **BACCustomer finden**

Mit der Funktion *FindCustomer* können Sie einen *BACCustomer* der *BACCustomers*-Auflistung anhand des Ländercodes (für Deutschland: *280*), der Bankleitzahl und der Benutzerkennung ermitteln. Die Funktion liefert den Index des gefundenen *BACCustomer*-Objekts zurück. Falls kein passender *BACCustomer* gefunden werden kann, liefert die Funktion den Wert -1:

```
Public Sub CustomerFinden()
   Dim objCustomer As BACCustomer
   Set objCustomer = GetBanking.Customers(0)
   Dim i As Integer
   i = GetBanking.Customers.FindCustomer(280. "70000997", "735160")
   Set objCustomer = GetBanking.Customers(i)
   Debug.Print objCustomer.BankCode
End Sub
```

Es gibt noch eine zweite Funktion, mit der Sie den Index eines *BACCustomer*-Objekts bestimmen können. Dabei handelt es sich um die *FindContact*-Methode. Diese erwartet den im Dialog *Banking Kontakte* in der ersten Spalte angegebenen Namen (siehe Abbildung 3.6). Ein Beispiel für diese Funktion sieht wie folgt aus:

```
Public Sub CustomerFinden_II()
   Dim objCustomer As BACCustomer
   Set objCustomer = GetBanking.Customers(0)
   Dim i As Integer
   i = GetBanking.Customers.FindContact("Commerzbank")
   Set objCustomer = GetBanking.Customers(i)
```

**Das BACBanking-Objekt** 

Debug.Print objCustomer.BankCode End Sub

dministrator für Homebanking Kontakte						
Folgende Homebanking Kontakte sind auf diesem System installiert. Die mit einem Ausrufezeichen markierten Kontakte müssen noch synchronisiert werden.						
Liste aller Kontakte:	-		Aktualisiere			
Kontakt	BLZ	Benutzerkennung				
Commerzbank	35040038	638415109800				
DataDesign Demoban	70000997	111130				
DataDasian Damahan	70000997	735160				
EEE Databesiyii Demobali		100100				
DataDesign Demoban	70000997	735161				
DataDesign Demoban	70000997 70000997	735161 735163				
DataDesign Demoban DataDesign Demoban DataDesign Demoban	70000997 70000997 70000997	735161 735163 735166				
DataDesign Demoban DataDesign Demoban DataDesign Demoban DataDesign Demoban	70000997 70000997 70000997 76010085	735161 735163 735166 937289858				

Abbildung 3.6: Kontakt-Name, der als Parameter für die FindContact-Funktion dient

### 3.3.5 BACCustomer löschen

Mit der Methode *DeleteCustomer* des *BACBanking*-Objekts können Sie einen der *BACCustomer* aus der Liste entfernen. Dazu übergeben Sie als Parameter den Ländercode (*280* für Deutschland), die Bankleitzahl und die Kontonummer an die Funktion *DeleteCustomer*:

```
Public Sub CustomerLoeschen()
    Debug.Print "Vor dem Löschen: " & GetBanking.Customers.count
    GetBanking.DeleteCustomer 280. "70000997". "735166"
    Debug.Print "Nach dem Löschen: " & GetBanking.Customers.count
End Sub
```

### Alle BACCustomer löschen

Sie können auch gleich alle *BACCustomer*-Objekte in einer Prozedur löschen. Dies erledigen Sie wie folgt:

```
Public Sub AlleCustomerLoeschen()
  Dim objCustomer As BACCustomer
  Do While GetBanking.Customers.count > 0
   Set objCustomer = GetBanking.Customers(0)
   With objCustomer
        GetBanking.DeleteCustomer .CountryCode, .BankCode, .UserID
   End With
```

Loop End Sub

Die Prozedur durchläuft so lange eine *Do While*-Schleife, bis die *Count*-Eigenschaft der *Customers*-Auflistung den Wert *0* liefert. Bis dahin wird in jedem Schleifendurchlauf der *BACCustomer* mit dem Index *0* aus der Liste gelöscht.

### 3.3.6 BACCustomer anlegen

Sie können auch per VBA einen neuen *BACCustomer* anlegen. Dazu verwenden Sie die Funktion *NewCustomer*. Diese erwartet drei Parameter: den Ländercode (*280* für Deutschland), die Bankleitzahl sowie die *UserID*. Die folgende Prozedur zeigt, wie dies funktioniert:

```
Public Sub CustomerAnlegen()
    Dim objCustomer As BACCustomer
    Set objCustomer = GetBanking.NewCustomer(280, "35040038", "637415109800")
End Sub
```

Wenn Sie danach den Dialog *Banking Kontakte* öffnen, erscheint dort allerdings ein Warnzeichen beim neuen Kontakt (siehe Abbildung 3.7).

ministrator für Homebanking Kontakte						
Folgende Homebanking Kontakte sind auf diesem System installiert. Die mit einem Ausrufezeichen markierten Kontakte müssen noch synchronisiert werden.						
Kontakt	BLZ	Benutzerkennung				
1 35040038:6374151	35040038	637415109800				

Abbildung 3.7: Ein per VBA-Code angelegter Kontakt

Was fehlt hier noch? Zum Beispiel sind für diesen noch keine Konten hinterlegt, es wurde kein Sicherheitsverfahren festgelegt, die HBCI/FinTS-Version ist nicht festgelegt und der Kontakt ist auch noch nicht synchronisiert. Kurzum: Um dies alles per Code zu realisieren, ist weiterer, nicht unerheblicher Aufwand erforderlich – was wir an dieser Stelle scheuen, da sich die Kontakte über die Systemsteuerung ja doch recht einfach anlegen lassen.

**Das BACBanking-Objekt** 

### 3.3.7 Customer synchronisieren

Manchmal scheitern Aktionen wie etwa das Abrufen des aktuellen Kontostandes daran, dass der Kontakt nicht synchronisiert ist. Dies können Sie mit der Funktion *SynchronizeCustomer* erledigen, die einen Verweis auf das *BACCustomer*-Objekt erwartet und den Wert *True* oder *False* als Ergebnis zurückliefert.

Wenn Sie beispielsweise einen per VBA frisch angelegten Kontakt synchronisieren wollen, erledigen Sie dies etwa wie folgt:

```
Public Sub CustomerAnlegenUndSynchronisieren()
   Dim objCustomer As BACCustomer
   Set objCustomer = GetBanking.NewCustomer(280, "35040038", "637415109800")
   Debug.Print GetBanking.SynchronizeCustomer(objCustomer)
End Sub
```

Dies löst, je nach Sicherheitsverfahren, etwa die Anzeige eines Dialogs zur Abfrage des PIN aus (siehe Abbildung 3.8).

📴 HBCI/FinTS-Kontakt	
Synchronisieren eines Homeban	king Kontakts
Für die gewünschte Funktion wird eine gültige PIN Homebanking Kontakt: 35040038:637415109800	l (Passwort) benötigt.
Schlüsseldatei: PIN (Passwort): Stellen Sie jetzt die Schlüsseldatei für den angegebenen Homebanking Kontakt bereit, geben Sie die zugehörige PIN (Passwort) an und klicken Sie dann auf Weiter'. Sicherheitshinweis: Die Schlüsseldatei sollte sicher verwahrt bleiben.	Auswählen 1 2 3 4 5 6 7 8 9 C 0 ←
Zurück Weiter	Abbrechen

Abbildung 3.8: Abfrage des PIN vor dem Synchronisieren des Kontaktes

Bei einem nur stückweise angelegten Konto wie oben erscheint nun die Meldung aus Abbildung 3.8.



Abbildung 3.9: Der Kontakt wurde nicht vollständig angelegt.

### 3.3.8 Optionen

Mit der *Options*-Eigenschaft des *BACBanking*-Objekts greifen Sie auf die in der Registry gespeicherten Anwendungsoptionen zu. Dazu übergeben Sie der *Options*-Eigenschaft jeweils den Namen der zu untersuchenden Option. Diese sind die wichtigsten Optionen:

- » *InstallDir*: Verzeichnis, in dem sich die *DDBAC*-Komponenten befinden, speziell die Komponente *DDBAC.DLL*.
- » DataDir: Verzeichnis zum Speichern der Datei DDUSERS.DAT, welche die mit der Systemsteuerung angelegten Daten enthält. Außerdem finden Sie dort die UPD- und BPD-Dateien.
- » PersistBPD: Legt fest, ob erhaltene BPD-Dateien in dem unter DataDir angegebenen Verzeichnis gespeichert werden sollen (1: BPD-Dateien speichern, 0: BPD-Dateien nicht speichern)
- » PersistUPD: Legt fest, ob erhaltene UPD-Dateien in dem unter DataDir angegebenen Verzeichnis gespeichert werden sollen (1: UPD-Dateien speichern, 0: UPD-Dateien nicht speichern)
- » *ResponseTimeout*: Gibt die Zeit an, wann bei einer Anfrage an einen Bankserver ein Timeout erfolgen soll. Standardwert ist *120.000* (120 Sekunden).
- » IsTraceOn: Gibt an, ob die durchgeführten Aktionen (gegebenenfalls mit Fehlerhinweisen) in einer Textdatei protokolliert werden sollen. Der Wert 1 aktiviert die Protokollierung, welche in einer Datei namens HBCILog.txt etwa im Verzeichnis C:\Users\<Benutzername>\ Documents erfolgt, 0 deaktiviert die Protokollierung.
- » Version: Liefert die Version der aktuell installierten DDBAC.dll, zum Beispiel 4.2.1.0.

Die folgende Prozedur gibt alle Eigenschaften mit Eigenschaftswerten im Direktfenster aus (auch solche, die in der obigen Auflistung nicht dokumentiert wurden):

```
Public Sub BankingOptions()
Debug.Print "InstallDir: " & GetBanking.Options("InstallDir")
Debug.Print "DataDir: " & GetBanking.Options("DataDir")
Debug.Print "PersistBPD: " & GetBanking.Options("PersistBPD")
```

#### **Das BACBanking-Objekt**

```
Debug.Print "PersistUPD: " & GetBanking.Options("PersistUPD")
Debug.Print "ResponseTimeout: " & GetBanking.Options("ResponseTimeout")
Debug.Print "IsTraceOn: " & GetBanking.Options("IsTraceOn")
Debug.Print "CardReaderClass2: " & GetBanking.Options("CardReaderClass2")
Debug.Print "CardTerminalID: " & GetBanking.Options("CardTerminalID")
Debug.Print "CardTerminalParameter: " & GetBanking.Options("CardTerminalParameter")
Debug.Print "CardTerminalProgID: " & GetBanking.Options("CardTerminalProgID")
Debug.Print "CustomerSystemStatusTAN: " ______ & GetBanking.Options("CustomizeUI")
Debug.Print "CustomizeUI: " & GetBanking.Options("CustomizeUI")
Debug.Print "CustomizeUI: " & GetBanking.Options("CustomizeUI")
Debug.Print "Version: " & GetBanking.Options("Version")
End Sub
```

Verschiedene Optionen können Sie auch per VBA anpassen. Wenn Sie beispielsweise wünschen, dass die Protokolldateien und die nachfolgend beschriebenen BPD- und UPD-Dateien etwa im gleichen Verzeichnis wie die Datenbankdatei gespeichert werden, von der aus Sie die Onlinebanking-Funktionen ausführen, rufen Sie die folgende Anweisung im Direktfenster auf:

```
GetBanking.Options("DataDir") = CurrentProject.Path
```

Dieser Pfad wird dann in der Registry wie in Abbildung 3.10 eingestellt.

Registrierungs-Editor			
Datei Bearbeiten Ansicht Favoriten ?			
Date:       Bearbeiten       Ansicht       Favoriten       ?         HKEY_CURRENT_USER       >       AppEvents       >       >       Console       >       >       >       Console       >       >       >       >       >       Console       >       >       >       >       >       Devisionment       >	Name ab (Standard) ab DataDir ab IsTraceOn	Typ REG_SZ REG_SZ REG_SZ	Daten (Wert nicht festgelegt) C:\Daten\Buecher\OnlinebankingMitAccess\Beispieldateien 1
AppDataLow Apple Inc. AVAST Software Classes Computer, Inc. AVAST Software Classes Cla	DDBAC		

Abbildung 3.10: Der neue Zielpfad für die Protokolldateien wird in der Registry vermerkt.

### 3.3.9 Optionen von Access aus einstellen

Damit Sie diese Optionen von Access aus bequem einstellen können, haben wir einen Optionen-Dialog namens *frmOptionen* zur Beispieldatenbank hinzugefügt. Dieser sieht wie in Abbildung 3.11 aus.

🗐 frmOptionen					23
Optionen					
Protokollverzeichnis:	C:\Users\Andre\A	ppData\Roaming\DataDesig	gn\DDBAC	:	
BPD speichern:	<b>V</b>				
UPD speichern:	$\checkmark$				
Timeout:	120000				
Protokollierung:	$\checkmark$				
DDBAC-Version:	5.3.23				
К					

Abbildung 3.11: Der Dialog frmOptionen

Beim Öffnen des Formulars sollen zunächst die entsprechenden Werte aus der Registry ausgelesen werden. Glücklicherweise bietet die *DDBAC*-Komponente ja die *Options*-Eigenschaft, mit der Sie auf alle Eigenschaftswerte zugreifen können. Die folgende Prozedur wird beim Laden des Formulars ausgelöst und füllt die einzelnen Steuerelemente, falls nötig:

```
Private Sub Form Load()
   Dim objBanking As BACBanking
   Dim bolBPD As Boolean
   Dim bolUPD As Boolean
   Dim bolProtokollierung As Boolean
   Dim lngTimeout As Long
   Set objBanking = GetBanking
   Me!txtProtokollverzeichnis = objBanking.Options("DataDir")
   If Len(objBanking.Options("PersistBPD")) = 0 Then
       bolBPD = True
   Else
       bolBPD = -(objBanking.Options("PersistBPD"))
   Fnd If
   Me!chkBPD = bolBPD
   If Len(objBanking.Options("PersistUPD")) = 0 Then
       bolUPD = True
   F1se
       bolUPD = -(objBanking.Options("PersistUPD"))
```

#### **Das BACBanking-Objekt**

```
Fnd If
   Me!chkUPD = bolUPD
   If Len(objBanking.Options("ResponseTimeout")) = 0 Then
        lngTimeout = 120000
   Else
       lngTimeout = objBanking.Options("ResponseTimeout")
   Fnd If
   Me!txtTimeout = lngTimeout
   If Len(objBanking.Options("IsTraceOn")) = 0 Then
       bolProtokollierung = False
   Else
       bolProtokollierung = -(objBanking.Options("IsTraceOn"))
   End If
   Me!chkProtokollierung = bolProtokollierung
   Me!txtDDBACVersion = objBanking.Options("Version")
End Sub
```

Zunächst füllt die Prozedur die Variable *objBanking* über die Funktion *GetBanking* mit einem Verweis auf ein neues oder bestehendes Objekt des Typs *BACBanking*. Dieses nutzt die Prozedur dann, um das aktuelle Protokollverzeichnis auszulesen (standardmäßig beispielsweise *C:\Users\<Benutzername>\AppData\Roaming\DataDesign\DDBAC*) und den Wert in das Textfeld *txtProtokollverzeichnis* zu schreiben.

Dann prüft die Prozedur, ob die Option *PersistBPD* bereits gefüllt ist. Falls nicht, wird der Standardwert (*True*) angenommen, sonst entweder der Wert *True* (für den Optionswert 1) oder *False* (für den Optionswert 0) in die Variable *bolBPD* eingetragen. Der Wert von *bolBPD* landet dann im Kontrollkästchen *chkBPD*. Für die Option *PersistUPD* verfährt die Prozedur analog.

Wenn die Option *ResponseTimeout* keinen Wert enthält, trägt die Prozedur den Standardwert *120.000* in das Textfeld *txtTimeout* ein, sonst den entsprechenden gefundenen Wert.

Schließlich fehlt noch die Option, ob eine Protokollierung erfolgen soll. Ist die Eigenschaft *IsTraceOn* leer, trägt die Prozedur den Standardwert *False* in die Variable *bolProtokollierung* ein, sonst verfährt die Prozedur wie bei *PersistBPD* und *PersistUPD*.

Schließlich trägt die Prozedur noch die Version der DDBAC-Komponente in das Textfeld *txtDDBAC-Version* ein (diese Eigenschaft kann natürlich nicht durch den Benutzer geändert werden).

Neben dem Textfeld zur Eingabe des Verzeichnisses für die Protokolldatei finden Sie eine Schaltfläche zum Auswählen des Verzeichnisses. Diese löst die folgende Prozedur aus:

```
Private Sub cmdVerzeichnisAuswaehlen_Click()
    Me!txtProtokollverzeichnis = OpenPathName(Nz(Me!txtProtokollverzeichnis, _
        CurrentProject.Path), "Verzeichnis auswählen")
End Sub
```

Kapitel 3 Grundlagen der DDBAC

Die hier verwendete Funktion OpenPathName finden Sie im Modul mdlDateifunktionen.

Die Schaltfläche cmdOK schließt das Formular:

```
Private Sub cmdOK_Click()
    DoCmd.Close acForm, Me.Name
End Sub
```

Damit löst diese Prozedur wie auch die anderen Wege, um das Formular zu schließen, das Ereignis *Form\_Close* aus. Dieses implementieren Sie wie folgt:

```
Private Sub Form Close()
   Dim objBanking As BACBanking
   Set objBanking = GetBanking
    If Not (Me!txtProtokollverzeichnis = objBanking.Options("DataDir")) Then
        objBanking.Options("DataDir") = Me!txtProtokollverzeichnis
    Fnd If
    If Len(obiBanking.Options("PersistBPD")) = 0 Then
        If Me!chkBPD = 0 Then
            objBanking.Options("PersistBPD") = 0
        End If
   Else
        objBanking.Options("PersistBPD") = -Me!chkBPD
    Fnd If
    If Len(obiBanking.Options("PersistUPD")) = 0 Then
        If Me!chkUPD = 0 Then
            objBanking.Options("PersistUPD") = 0
        End If
    F1se
        objBanking.Options("PersistUPD") = -Me!chkUPD
    Fnd If
    If Len(objBanking.Options("ResponseTimeout")) = 0 Then
        If Not Me!txtTimeout = 120000 Then
            objBanking.Options("ResponseTimeout") = Me!txtTimeout
        Fnd If
    Else
        objBanking.Options("ResponseTimeout") = Me!txtTimeout
    End If
    If Len(objBanking.Options("IsTraceOn")) = 0 Then
        If Me!chkProtokollierung = True Then
            objBanking.Options("IsTraceOn") = 1
        Fnd If
    F1se
```

**Das BACBanking-Objekt** 

```
objBanking.Options("IsTraceOn") = -Me!chkProtokollierung
End If
Fnd Sub
```

Die Prozedur füllt wieder eine Variable namens *objBanking* mit einem Verweis auf das *BACBanking*-Objekt. Im ersten Schritt vergleicht sie dann das aktuell in *txtProtokollverzeichnis* enthaltene Verzeichnis mit dem der Option *DataDir*. Wenn sich beide unterschieden, trägt die Prozedur den Wert aus dem Textfeld für die entsprechende Option ein.

Dann sind die beiden Optionen *PersistBPD* und *PersistUPD* an der Reihe. Ist die Option noch nicht in der Registry angelegt, liefert *objBanking.Options("PersistBPD")* eine leere Zeichenkette. *PersistBPD* hat aber den Standardwert 1, also *True*, und soll dementsprechend nur angelegt werden, wenn der Benutzer im Formular den Wert *False* eingestellt hat. Ist *PersistBPD* hingegen schon angelegt, soll auf jeden Fall der im Formular festgelegte Wert eingetragen werden beziehungsweise im Falle von *True* (-1) der Wert 1.

*ResponseTimeout* hat den Standardwert *120.000*. Wenn die Option noch nicht in der Registry angelegt wurde, soll dies daher auch nur geschehen, wenn der Benutzer einen Wert ungleich *120.000* eingestellt hat. Wurde der Wert ohnehin bereits in der Registry gespeichert, soll er einfach mit dem aktuellen Wert aus dem Textfeld *txtTimeout* überschrieben werden.

Die Eigenschaft *IsTraceOn* ist standardmäßig nicht aktiviert (auch wenn sie nicht in der Registry vermerkt ist). Daher erfolgt auch hier die Prüfung, ob die entsprechende Option bereits gespeichert wurde. Falls nicht, liefert *Len(objBanking.Options("IsTraceOn"))* den Wert 0. Deshalb soll die Option, soweit noch nicht vorhanden, nur eingestellt werden, wenn der Benutzer diese im Formular aktiviert hat.

### 3.3.10 BPD und UPD

Die oben erwähnten *BPD*- und *UPD*-Dateien enthalten Informationen über die Bank und den Account. Sie werden nach dem Synchronisieren des Banking-Kontaktes von der Bank ermittelt und in dem Verzeichnis gespeichert, das auch die Datei *ddusers.dat* aufnimmt (siehe Abbildung 3.12).

💽 🗢 🖟 « AppData 🕨 Roam	ning	► DataDesign ► DDBAC	✓ 4→ DDBAC d	urchsuchen	
Organisieren 🔻 🛛 In Bibliothek au	ıfnel	nmen 🔻 Freigeben für 🔻 Brennen	Neuer Ordner		• 🔟 🔞
	*	Name	Änderungsdatum	Тур	Größe
Computer		280_35040038_638415109800_300.bpd	06.04.2014 13:23	BPD-Datei	6 KB
BOUTCAMP (C:)		280_35040038_638415109800_300.spa	06.04.2014 13:23	SPA-Datei	2 KB
Andre A		280_35040038_638415109800_300.upd	06.04.2014 13:23	UPD-Datei	2 KB
	E	ddusers.dat	06.04.2014 13:23	DAT-Datei	1 KB
Roaming					
🔒 Adobe					
🌗 Apple Computer					
🌗 AVAST Software					
\mu chc.4875E02D9FB21					
🍌 DAEMON Tools Lite					
🍌 DataDesign					
DDBAC	Ŧ				

Abbildung 3.12: Speicherort von UPD-, BPD- und SPA-Datei

#### Bankparameterdaten

Die BPD-Datei enthält die Bankparameterdaten. Darunter befinden sich beispielsweise die BLZ, der Name der Bank und die Serveradresse sowie die möglichen Sicherheitsmechanismen. Außerdem finden Sie hier Informationen, welche Geschäftsvorfälle die Bank unterstützt beziehungsweise welche Segmente. Zu den Segmenten liefert die Datei weitere Detailinformationen wie etwa die Anzahl von Verwendungszweckzeilen bei herkömmlichen Überweisungen.

Nachfolgend finden Sie etwa die ersten beiden Zeilen dieser Datei für ein Postbank-Konto:

```
HIBPA:6:3:4+7+280:76010085+Postbank Nürnberg+0+1+220:300+9999'
HIKOM:7:4:4+280:76010085+1+3:https?://hbci.postbank.de/banking/hbci.do::MIM:1'
```

Bei der Commerzbank sehen diese Zeilen ähnlich aus:

```
HIBPA:8:3:4+83+280:35040038+COMMERZBANK 35040038+0+1+201:210:220:300+2126'
HIKOM:9:4:4+280:35040038+1+2:hbci.commerzbank.de'
```

Weiter unten gehen wir im Detail darauf ein, wie Sie für spätere Anwendungen die Details der Bankparameterdaten auslesen können.

#### Userparameterdaten

Die UPD-Datei enthält die Userparameterdaten. Dazu gehören die Benutzerkennung, die Kontonummer, die Produktbezeichnung und die Auflistung der Segmente.

**Das BACCustomer-Objekt** 

#### Hier die ersten Zeilen einer UPD-Datei für die Postbank:

```
HIUPA:63:4:4+937289811+96+0+Andre Minhorst'
HIUPD:0:6:4+250544111::280:60010070+DE07600100700250544111+937289811+1+EUR+Andre
Minhorst++Postbank Giro plus++
```

#### Und hier für die Commerzbank:

```
HIUPA:121:4:4+638415109111+82+0'
HIUPD:0:6:4+411890222:EUR100:280:35040038++638415109111++EUR+ANDRE
MINHORST++Kontokorrent++
```

## 3.4 Das BACCustomer-Objekt

Das *BACCustomer*-Objekte haben Sie ja teilweise bereits kennengelernt – zumindest haben Sie neue Exemplare mit dem entsprechenden *DDBAC*-Dialog über die Systemsteuerung (erfolgreich) und manuell (weniger erfolgreich) erstellt und die Kontakte wieder gelöscht. Für weitere Aktionen verwenden wir jedoch das *BACContact*-Objekt, das dem *BACCustomer*-Objekt ähnelt, aber mehr Möglichkeiten bietet.

# 3.5 Das BACContact-Objekt

Das *BACContact*-Objekt repräsentiert einen Eintrag des Dialogs *Administrator für Homebanking Kontakte* (siehe Abbildung 3.13).

👜 Administrator für Homebanking	Kontakte		×
Homebanking Kontakte Herstellerh	nweis		
Kontakte Folgende Homebankir einem Ausrufungszeic werden.	g Kontakte sind auf dies en markierten Kontakte	em System installiert. Die mit müssen noch synchronisiert	_
Kontakt	BLZ	Benutzerkennung	
Commerzbank	35040038 TS3 70000997 76010085		

Abbildung 3.13: BACContact-Objekte im Administrator für Homebanking Kontakte

Über das *BACContact*-Objekt erhalten Sie Zugriff auf die Eigenschaften des Kontaktes, der Bank und über die *Accounts*-Auflistung auch auf die einzelnen Konten des Kontakts. Der *BACContact* wird über die Eigenschaft *Fields("ID")* eindeutig identifiziert. Dieses Feld ent-

hält einen Eintrag in der Form *<Ländercode>:<Bankleitzahl>:<Benutzerkennung>*, also etwa 280:12345678:9876543210.

Das BACContact-Objekt bietet die folgenden Eigenschaften an:

- » Property AccountData As BACAccountData [r/o]
- » Property Accounts As BACAccounts [r/o]
- » Property BankCode As String [r/o]
- » Property BankData As BACBankData [r/o]
- » Property CountryCode As Long [r/o]
- » Property Fields(ByVal sName As String) As String [r/w] [default]
- » Property UserID As String [r/o]

Außerdem offeriert es die folgenden Funktionen und Methoden:

- » Function ActivateTANList(ByVal hWnd As Long) As BACDialogResults
- » Function Authenticate(ByVal sPassphrase As String) As IBACSecurity
- » Function AuthenticateUI() As IBACSecurity
- » Function CanActivateTANList() As Boolean
- » Function CanChangeKeys() As Boolean
- » Function CanChangePin() As Boolean
- » Function CanChangeTANMethod() As Boolean
- » Function CanGenerateIniLetter() As Boolean
- » Function CanLockKeys() As Boolean
- » Function CanLockPIN() As Boolean
- » Function CanLockTANList() As Boolean
- » Function CanRequestTANList() As Boolean
- » Function CanShowTANList() As Boolean
- » Function CanUnlockPIN() As Boolean
- » Function CanUpgradeKeys() As Boolean
- » Function ChangeConnection(ByVal hWnd As Long) As BACDialogResults
- » Function ChangeHbciVersion(ByVal hWnd As Long) As BACDialogResults

#### Das BACContact-Objekt

- » Function ChangeKeys(ByVal hWnd As Long) As BACDialogResults
- » Function ChangePin(ByVal hWnd As Long) As BACDialogResults
- » Function ChangeTANMethod(ByVal hWnd As Long) As BACDialogResults
- » Function ChangeUserId(ByVal hWnd As Long) As BACDialogResults
- » Sub Create()
- » Sub Delete()
- » Function Edit(ByVal hWnd As Long) As BACDialogResults
- » Function Execute(ByVal hWnd As Long, ByVal bsRequest As String) As BACDialogResults
- » Function GenerateIniLetter(ByVal hWnd As Long) As BACDialogResults
- » Function GetCustomerSystemID(ByVal piSecurity As IBACSecurity) As String
- » Function GetSignatureID(ByVal piSecurity As IBACSecurity) As String
- » Function IsPersistant() As Boolean
- » Sub Load(ByVal bstrID As String)
- » Function LockKeys(ByVal hWnd As Long) As BACDialogResults
- » Function LockPIN(ByVal hWnd As Long) As BACDialogResults
- » Function LockTANList(ByVal hWnd As Long) As BACDialogResults
- » Function NewDialog(ByVal sPassphrase As String) As BACDialog
- » Function NewDialogUI() As BACDialog
- » Function NewWizard(ByVal hWnd As Long) As BACDialogResults
- » Function QueryPin() As String
- » Function RequestTANList(ByVal hWnd As Long) As BACDialogResults
- » Sub Save()
- » Sub SetCustomerSystemID(ByVal piSecurity As IBACSecurity, ByVal sCustomerSystemID As String)
- » Function ShowTANList(ByVal hWnd As Long) As BACDialogResults
- » Sub SignMessage(ByVal piMessage As BACMessage, ByVal sPassphrase As String, [ByVal nSignerRole As BACSignerRole = bacSignerConfirms (3)])
- » Sub SignMessageUI(ByVal piMessage As BACMessage, [ByVal nSignerRole As BACSignerRole = bacSignerConfirms (3)])

- » Function Synchronize(ByVal hWnd As Long) As BACDialogResults
- » Function TransactionNeedsTAN(ByVal sSegmentType As String) As Long
- » Function UnlockPIN(ByVal hWnd As Long) As BACDialogResults
- » Sub UpdateAccountInformation(ByVal piDialog As BACDialog)
- » Function UpgradeKeys(ByVal hWnd As Long) As BACDialogResults

Die wichtigsten Elemente erläutern wir in den folgenden Abschnitten.

### 3.5.1 Zugriff auf BACContact-Objekte

Wenn Sie auf ein solches *BACContact*-Objekt zugreifen möchten, deklarieren Sie zunächst eine entsprechende Objektvariable:

Dim objContact As BACContact

Dann benötigen Sie eine Auflistung des Typs *BACContacts*, um auf einen der Einträge zuzugreifen. Diese Auflistung finden Sie im Gegensatz zur *BACAccounts*-Auflistung nicht als Eigenschaft des *BACBanking*-Objekts, sondern Sie müssen diese als neues Objekt erstellen und dann füllen. Dies geschieht beispielsweise wie in den folgenden drei Anweisungen:

```
Dim objContacts As BACContacts
Set objContacts = New BACContacts
objContacts.Populate ""
```

Die erste Anweisung deklariert das Objekt, die zweite erstellt es neu und die dritte verwendet die *Populate*-Methode, um die Auflistung mit den Daten aus der Datei *ddusers.dat* zu füllen.

Dann greifen Sie über die Auflistung *objContacts* auf die einzelnen *BACContact*-Objekte zu. Im folgenden Beispiel referenzieren wir mit der Variablen *objContact* das erste Element der Auflistung und geben die drei Eigenschaften *BankCode* (BLZ), *CountryCode* (Länderkennzeichen) und *UserID* (Benutzerkennung) im Direktbereich des VBA-Editors aus:

```
Set objContact = objContacts(0)
Debug.Print objContact.BankCode
Debug.Print objContact.CountryCode
Debug.Print objContact.UserID
```

Sie können auch in einer Schleife alle BACCustomer-Objekte durchlaufen:

```
Public Sub BACContactsSchleife()
  Dim objContact As BACContact
  Dim objContacts As BACContacts
  Set objContacts = New BACContacts
  objContacts.Populate ""
```

Das BACContact-Objekt

```
For Each objContact In objContacts
    Debug.Print objContact.BankCode, objContact.CountryCode, objContact.UserID
    Next objContact
End Sub
```

### 3.5.2 Vereinfachung für den Zugriff auf die BACContacts-Auflistung

Da wir in den folgenden Abschnitten noch häufiger auf die Auflistung des Typs *BACContacts* zugreifen wollen, erstellen wir eine kleine Funktion ähnlich wie *GetBanking*, um diese schnell zu füllen.

Zunächst deklarieren Sie dazu eine entsprechende Objektvariable namens *m\_Contacts*, welche die jeweils aktuelle Auflistung enthält (siehe Modul *mdlOnlinebanking\_Objekte*):

Private m Contacts As BACContacts

Die folgende Funktion erwartet einen optionalen Parameter:

» bolReset: gibt an, ob die Kontakte neu eingelesen werden sollen

Die Funktion prüft zunächst, ob *m\_Contacts* bereits einmal erstellt wurde. Falls nicht, erstellt die Funktion die Auflistung neu und füllt sie mit der *Populate-*Methode, wobei der in *strFilter* enthaltene Ausdruck als Filter verwendet wird (standardmäßig eine leere Zeichenkette). Falls *m\_Contacts* schon vorhanden ist, prüft die Prozedur den Parameter *bolReset*. Ist dieser *True*, füllt die Funktion *m\_Contacts* neu mit der *Populate-*Methode und dem angegebenen Filterausdruck. Der Verweis auf *m\_Contacts* wird dann als Funktionswert zurückgegeben:

```
Public Function GetContacts(Optional bolReset As Boolean) As BACContacts
    If m_Contacts Is Nothing Then
        Set m_Contacts = New BACContacts
        m_Contacts.Populate ""
    Else
        If bolReset = True Then
            m_Contacts.Populate ""
        End If
    End If
    Set GetContacts = m_Contacts
End Function
```

### 3.5.3 BACContact-Fields

Das BACContact-Objekt bietet mit der Fields-Eigenschaft die Möglichkeit, auf verschiedene in der ddusers.dat-Datei gespeicherte Informationen zuzugreifen. Die folgende Prozedur gibt bei-

#### Kapitel 3 Grundlagen der DDBAC

spielhaft alle aktuell verwendeten Eigenschaften samt Eigenschaftsname im Direktbereich des VBA-Editors aus:

```
Public Sub BACContactFields()
   Dim objContacts As BACContacts
   Dim objContact As BACContact
   Set objContacts = New BACContacts
   objContacts.Populate ""
   Set obiContact = obiContacts(0)
   With objContact
       Debug.Print "Contact", .Fields("Contact")
       Debug.Print "SecurityProgID", .Fields("SecurityProgID")
       Debug.Print "SecurityMediaID", .Fields("SecurityMediaID")
       Debug.Print "SecurityMediaType", .Fields("SecurityMediaType")
       Debug.Print "CustomerSystemStatus", .Fields("CustomerSystemStatus")
       Debug.Print "BankName". .Fields("BankName")
       Debug.Print "CustomerID", .Fields("CustomerID")
       Debug.Print "BankUserID", .Fields("BankUserID")
       Debug.Print "CommunicationsService", .Fields("CommunicationsService")
       Debug.Print "CommunicationsAddress", .Fields("CommunicationsAddress")
       Debug.Print "CommunicationsAddressSuffix". .Fields("CommunicationsAddressSuffix")
       Debug.Print "CustomerSystemID", .Fields("CustomerSystemID")
       Debug.Print "SignatureID", .Fields("SignatureID")
       Debug.Print "HBCIVersion". .Fields("HBCIVersion")
       Debug.Print "NeedSynchronisation", .Fields("NeedSynchronisation")
       Debug.Print "ManualUPD", .Fields("ManualUPD")
       Debug.Print "UPDVersion", .Fields("UPDVersion")
       Debug.Print "BPDVersion". .Fields("BPDVersion")
       Debug.Print "ID", .Fields("ID")
       Debug.Print "HKVVBVersion", .Fields("HKVVBVersion")
       Debug.Print "ManualITan", .Fields("ManualITan")
       Debug.Print "ITanSupported", .Fields("ITanSupported")
       Debug.Print "ITANVerfahren", .Fields("ITANVerfahren")
       Debug.Print "Sicherheitsfunktion", .Fields("Sicherheitsfunktion")
       Debug.Print "ProductName", .Fields("ProductName")
       Debug.Print "ProductVersion", .Fields("ProductVersion")
       Debug.Print "SecurityMediaID", .Fields("SecurityMediaID")
       Debug.Print "SecurityMediaDescription", .Fields("SecurityMediaDescription")
       Debug.Print "MinPinLength", .Fields("MinPinLength")
       Debug.Print "MaxPinLength", .Fields("MaxPinLength")
       Debug.Print "PinOnlyNumeric", .Fields("PinOnlyNumeric")
       Debug.Print "PinRequirements", .Fields("PinRequirements")
```

#### Das BACContact-Objekt

```
Debug.Print "CanChangeKeys", .Fields("CanChangeKeys")
Debug.Print "CanUpgradeKeys", .Fields("CanUpgradeKeys")
Debug.Print "RemainingSignatures", .Fields("RemainingSignatures")
End With
End Sub
```

### 3.5.4 Die AccountData-Eigenschaft

Diese Eigenschaft erlaubt den Zugriff auf die einzelnen Accounts des *BACCustomer*-Objekts. Weiter oben haben Sie erfahren, dass alle Account-Daten in der *UPD*-Datei gespeichert werden. Auf die einzelnen Accounts können Sie dabei über die *AccountData*-Eigenschaft zugreifen, wobei Sie den Index des gewünschten Eintrags als Parameter angeben. Die folgende Prozedur speichert beispielsweise die Daten des ersten Accounts in einer Textdatei namens *Accountdata.txt*:

```
Public Sub BACContactAccountData()
   Dim objContacts As BACContacts
   Dim objContact As BACContact
   Set objContacts = New BACContacts
   objContacts.Populate ""
   Set objContact = objContacts(0)
   objContact.AccountData(0).SaveAs CurrentProject.Path & "\Accountdata.txt"
End Sub
```

Der Inhalt dieser Datei entspricht dabei einem Eintrag in der UDP-Datei (sofern diese mehr als ein Konto enthält). Sie können auch die Eigenschaften zu jedem einzelnen Account im XML-Format ausgeben lassen. Dazu verwenden Sie die *GetXML*-Methode zusammen mit *Debug.Print*:

```
Public Sub BACContactAccountDataXML()
   Dim objContact As BACContact
   Dim objAccountData As BACAccountData
   Dim objSegment As BACSegment
   Set objContact = GetContacts.Item(0)
   Set objAccountData = objContact.AccountData
   For Each objSegment In objAccountData.Segments
        Debug.Print objSegment.GetXML
   Next objSegment
```

```
End Sub
```

Die *GetXML*-Methode steht allerdings nur zur Verfügung, wenn Sie eine Variable des Typs *BAC-AccountData* verwenden. Das Ergebnis sieht etwa wie folgt aus und liefert die Basisdaten des Kontos sowie die erlaubten Geschäftsvorfälle:

```
<SEGMENT TYPE="HIUPD" VERSION="6" REFSEQNO="4">
<Kontoverbindung>
```

```
<Kontonummer Format="Id">937289123</Kontonummer>
    <Laenderkennzeichen Format="country code">280</Laenderkennzeichen>
    <Kreditinstitutcode Format="alphanumeric">76010085</Kreditinstitutcode>
  </Kontoverbindung>
  <IBAN Format="alphanumeric">DE57760100850937289123</IBAN>
  <Kundenid Format="Id">937289123</Kundenid>
  <Kontoart Format="numeric">1</Kontoart>
  <Kontowaehrung Format="currency">EUR</Kontowaehrung>
  <NameEins Format="alphanumeric">Andre Minhorst</NameEins>
  <Kontoproduktbezeichnung Format="alphanumeric">Postbank Giro plus
    </Kontoproduktbezeichnung>
  <ErlaubteGeschaeftsvorfaelle>
    <Geschaeftsvorfall Format="alphanumeric">HKSPA</Geschaeftsvorfall>
    <MinAnzahlSignaturen Format="numeric">1</MinAnzahlSignaturen>
  </FrlaubteGeschaeftsvorfaelle>
  <ErlaubteGeschaeftsvorfaelle>
    <Geschaeftsvorfall Format="alphanumeric">DKTSP</Geschaeftsvorfall>
    <MinAnzahlSignaturen Format="numeric">1</MinAnzahlSignaturen>
  </ErlaubteGeschaeftsvorfaelle>
</SEGMENT>
```

### 3.5.5 Die BankData-Eigenschaft

Mit der *BankData*-Eigenschaft, wie wiederum ein Objekt des Typs *BACBankData* zurückliefert, können Sie zunächst einfache Eigenschaften der Bank abfragen:

```
Public Sub BACContactBankData()
  Dim objContact As BACContact
  Dim objBankData As BACBankData
  Set objContact = GetContacts.Item(0)
  Set objBankData = objContact.BankData
  With objBankData
   Debug.Print .BankCode
   Debug.Print .CountryCode
  End With
End Sub
```

Außerdem erhalten Sie damit Zugriff auf alle möglichen Segmente. Das *BACSegment*-Objekt beschreiben wir weiter hinten, hier erstmal eine Möglichkeit, die fünf Buchstaben langen Zeichenfolgen, die als Kennzeichnung der Segmente dienen, sowie ihre Version auszugeben:

```
Public Sub BACContactBankDataSegmente()
```

#### Das BACContact-Objekt

```
Dim objContact As BACContact
Dim objSegment As BACSegment
Set objContact = GetContacts.Item(0)
For Each objSegment In objContact.BankData.Segments
Debug.Print objSegment.Type, objSegment.Version
Next objSegment
End Sub
```

### 3.5.6 Die Accounts-Auflistung

Über die *Accounts*-Auflistung greifen Sie auf die zu einem *BACContact*-Objekt gehörenden Konten zu. Im folgenden Beispiel geben wir dabei in einer *For Each*-Schleife die Kontonummer des jeweiligen Kontos aus:

```
Public Sub BACAccounts()
   Dim objContact As BACContact
   Dim objAccount As BACAccount
   Set objContact = GetContacts.Item(0)
   For Each objAccount In objContact.Accounts
        Debug.Print objAccount.AccountNumber
   Next objAccount
End Sub
```

Dies lässt sich natürlich auch noch mit einer *For Each*-Schleife über alle *BACContact*-Elemente kombinieren, sodass Sie alle Kontakte mit ihren Konten ausgeben können:

```
Public Sub BACContactsUndAccounts()
Dim objContact As BACContact
Dim objAccount As BACAccount
Debug.Print "Contacts: "
For Each objContact In GetContacts
Debug.Print "BLZ: " & objContact.BankCode & "UserID: " & objContact.UserID
Debug.Print " Accounts:"
For Each objAccount In objContact.Accounts
Debug.Print " " & objAccount.AcctName
Next objAccount
Next objContact
```

End Sub

Die *Accounts*-Auflistung bietet mit der *Count*-Eigenschaft auch noch die Möglichkeit, die Anzahl der Accounts auszugeben.

### 3.5.7 BACContact-Objekte per Kombinationsfeld

In vielen Beispielen in den folgenden Kapiteln werden Sie über ein Kombinationsfeld auf die *BACContact*-Objekte zugreifen, die Sie mit dem *Administrator für Homebanking Kontakte* angelegt haben. Dieses legen Sie im Formular wie in Abbildung 3.14 an.

frmBeispieleBACContact 📼	) 🗆	23
• · · · 1 · · · 2 · · · 3 · · · 4 · · · 5 · · · 6 · · · 7 · · · 8 · · · 9 · · · 10 ·	1 - 11 - 1	• 12 📥
Formularkopf		
Beispiele BACContact		
✓ Detailbereich		
BACContact Ungebunden	-	
	-	
13 Create		
Formularfuß		_
СК		

Abbildung 3.14: Kombinationsfeld zur Auswahl von BACContact-Objekten per Kombinationsfeld

In der Entwurfsansicht soll das Kombinationsfeld die wichtigsten Informationen zum jeweiligen Bankkontakt anzeigen, in diesem Fall die benutzerdefinierte Bezeichnung, die Bankleitzahl sowie die *UserID* (siehe Abbildung 3.15).

	😑 Beispiele	zum BACContact-Objekt		-		23
	8	Beispiele BACContact				
	BACContact:	Postbank 937				-
		Postbank 937	_			
	EOActivat	Commerzbank			15	- 1
	Change					- 1
. 0	- 4 - 1	and the second sec	الد المراجع		<u> </u>	المراجعة
1						1
	🔀 ОК					

Abbildung 3.15: Auswahl eines BACContact-Objekts

Damit dies geschieht, sind zwei Schritte nötig:

#### Das BACContact-Objekt

- » Einstellen der Eigenschaft Herkunftsart des Kombinationsfelds auf den Wert Wertliste
- » Füllen des Kombinationsfeldes mit einer speziell dafür vorgesehen Funktion namens Get-ContactList

Dies soll gleich beim Öffnen des Formulars geschehen. Dort lösen wir die folgende Prozedur aus, welche die beiden benötigten Schritte ausführt und wie folgt aussieht:

```
Private Sub Form_Open(Cancel As Integer)
Me!cboContacts.RowSourceType = "Value List"
Me!cboContacts.RowSource = GetContactList(True)
If Len(Me!cboContacts.RowSource) = 0 Then
MsgBox "Es wurden noch keine Kontakte angelegt." & vbCrLf & vbCrLf _
& "Dies können Sie über den Eintrag 'Homebanking Kontakte' " _
& "in der Systemsteuerung erledigen."
Cancel = True
Exit Sub
End If
Me!cboContacts = Me!cboContacts.ItemData(0)
End Sub
```

Die Prozedur führt außerdem noch eine Abfrage der Anzahl der Einträge in das Kombinationsfeld *cboContacts* aus. Sollte diese die Anzahl *O* zurückliefern, hat der Benutzer noch keine Kontakte angelegt – in diesem Fall soll das Formular unter Ausgabe einer entsprechenden Meldung gleich wieder geschlossen werden.

Die Funktion *GetContactList* finden Sie im Modul *mdlOnlinebanking* der Beispieldatenbank. Diese Prozedur erwartet mit *bolReset* einen optionalen *Boolean*-Wert, der angibt, ob die Kontakte neu eingelesen werden sollen oder ob, sofern bereits geschehen, die bereits vorhandene Liste der *BACContact*-Elemente verwendet werden soll. Dies ist sinnvoll, wenn Sie während der Entwicklung des öfteren Kontakte hinzufügen oder entfernen.

Die Funktion durchläuft alle Elemente des Typs *BACContact* der Auflistung, die durch eine weitere Funktion namens *GetContacts* geliefert wird. Dieser geben Sie den Parameter *bolReset* weiter, die Funktion sehen wir uns weiter unten an.

Die zurückgelieferten *BACContact*-Objekte werden in einer *For Each*-Schleife durchlaufen. Dabei setzt die Funktion *GetContactList* in der String-Variablen *strContacts* einen Ausdruck zusammen, der jeweils aus dem Wert einer Zählervariablen namens *i* (als Index), der *UserID*, der Bankleitzahl und einer aus dem Kontaktnamen, der Bankleitzahl und der *UserID* bestehenden Zeichenkette resultiert. Das Ergebnis wird dann als Funktionswert zurückgegeben:

```
Public Function GetContactList(Optional bolReset As Boolean) As String
Dim objContact As BACContact
Dim strContacts As String
```

```
Dim i As Integer
For Each objContact In GetContacts(bolReset)
    strContacts = strContacts & i & ":" & objContact.UserID & ":" _
        & objContact.BankCode & ":" _
        & objContact("Contact") _
        & "|" & objContact.BankCode _
        & "|" & objContact.UserID & ":"
        i = i + 1
    Next objContact
    GetContactList = strContacts
End Function
```

Ein Beispiel für diese Zeichenkette ist der folgende Ausdruck:

0:937289123:76010123:Postbank 937|76010123|937289123: 1:638415109123:35040123:Commerzbank|35040123|638415109123: 2:735123:70000123:DataDesign Demobank FinTS3|70000123|735123:

Die Funktion *GetContacts* haben Sie bereits weiter oben unter »Vereinfachung für den Zugriff auf die BACContacts-Auflistung«, Seite 65 kennengelernt.

### 3.5.8 Dialoge für die Verwaltung eines Kontakts

Das BACContact-Objekt liefert eine ganze Reihe von Dialogen, mit denen Sie verschiedene Aktionen durchführen können. Diese Dialoge öffnen Sie über entsprechende Methoden des BACContact-Objekts. Da die meisten davon ein gültiges Windows-Handle als Parameter erwarten, finden Sie die Beispiele in einem Formular namens frmBeispieleBACContact.

Das Formular bietet ein Kombinationsfeld, das alle aktuell verfügbaren *BACContact*-Elemente zur Auswahl offeriert. Wie dieses gefüllt wird, beschreiben wir ausführlich in *»Konten per VBA verwalten«, Seite 135*.

#### Neue TAN-Liste aktivieren

Diesem Formular fügen wir zunächst eine Schaltfläche namens *cmdActivateTANList* hinzu (siehe Abbildung 3.16).

#### Das BACContact-Objekt

-8	frmBeispieleBACContact 🛛 🗆 🗉	23
	······································	
	✓ Formularkopf	
- - 1 -	Beispiele BACContact	_
	Octailbereich	
1	BACContact: Ungebunden	
1	ActivateTANList	
2	ОК	
	✓ Formularfuß	
		<ul><li>▼</li><li>▶</li></ul>

Abbildung 3.16: Formular mit einer Schaltfläche zum Öffnen eines Dialogs

Die Prozedur, die beim Anklicken dieser Schaltfläche ausgelöst wird, sieht so aus:

```
Private Sub cmdActivateTANList_Click()
   Dim objContact As BACContact
   Dim intResults As BACDialogResults
   Set objContact = GetContacts.Item(0)
   intResults = objContact.ActivateTANList(Me.Hwnd)
   Select Case intResults
      Case bacDialogResultCancelled
      MsgBox "Der Dialog wurde abgebrochen."
   Case bacDialogResultOK
      MsgBox "Der Dialog wurde erfolgreich beendet."
   End Select
End Sub
```

Die Funktion *ActivateTANList* ruft den Dialog aus Abbildung 3.17 auf. Das Ergebnis der Funktion landet in der Variablen *intResults* vom Typ *BACDialogResults*. Das Ergebnis kann zwei Werte annehmen:

- » *bacDialogResultCancelled*: Der Dialog wurde vom Benutzer abgebrochen.
- » *bacDialogResultOK*: Der Dialog wurde erfolgreich beendet.

HBCI/FinTS-Kontakt	<b>—</b> ×
Aktivieren einer neuen TAN List	e
Für die gewünschte Funktion wird eine gültige f	PIN (Passwort) benötigt.
Homebanking Kontakt: Postbank 937	
TAN Verfahren:	Verfügbare TAN Medien:
901 mobileTAN 👻	Mobile TAN: mT:iphone 🕞
PIN (Passwort):	
	4 5 6
	789

Abbildung 3.17: Dialog zum Aktivieren einer neuen TAN-Liste

Wenn der Benutzer des mit *BACContact* referenzierten Kontaktes seine TAN-Nummern von einer TAN-Liste bezieht, benötigt er eine neue Liste, wenn alle TANs der alten Liste verbraucht sind. In diesem Fall kann er die neue Liste mit dem hier gezeigten Dialog freischalten.

#### Prüfen, ob TAN-Liste aktiviert werden kann

Aber macht diese Funktion überhaupt Sinn, wenn der Kontakt beispielsweise nur mit mobilen TAN-Nummern arbeitet? Nein, macht sie nicht. Deshalb gibt es eine weitere Funktion, mit der Sie prüfen können, ob überhaupt eine neue TAN-Liste aktiviert werden kann. Diese verwenden wir in einer weiteren Ereignisprozedur, die diesmal durch die Schaltfläche *cmdCanActivateTAN-List* ausgelöst wird und wie folgt aussieht:

```
Private Sub cmdCanActivateTANList_Click()
  Dim objContact As BACContact
  Dim bolCanActivateTANList As Boolean
  Dim intResults As BACDialogResults
  Set objContact = GetContacts.Item(0)
  bolCanActivateTANList = objContact.CanActivateTANList
  If bolCanActivateTANList = True Then
    Me!cmdActivateTANList.Enabled = True
    MsgBox "TAN-Liste kann für diesen Kontakt aktiviert werden."
  Else
    Me!cmdActivateTANList.Enabled = False
```
# 4 Konten per VBA verwalten

Dieses Kapitel zeigt, wie Sie die über die Benutzeroberfläche ermittelten Konten per VBA einlesen und für weitere Aktionen nutzen können.

# 4.1 Kontakt herstellen

Wenn Sie das *DDBAC Software Development Kit* installiert und ein oder mehrere Kontakte eingerichtet haben, können Sie gleich per VBA auf die eingerichteten Daten zugreifen. Das ist wichtig, weil Sie ja im Kontext eines bestimmten Benutzers und eines Bankkontos arbeiten möchten, um etwa Kontostände oder Umsätze einzulesen oder auch eine Überweisung zu tätigen.

Sie müssen also zunächst einmal das Benutzerkonto (später abgebildet durch das *Contact*-Objekt) und das Bankkonto (entspricht dem *Account*-Objekt) auswählen, mit dem Sie arbeiten möchten. Grundlage dafür ist ein Formular, das zwei abhängige Kombinationsfelder zur Auswahl der gewünschten Einträge anbietet.

Erstellen Sie also ein neues, leeres Formular, speichern Sie es unter dem Namen *frmOnlinebanking* und öffnen Sie es in der Entwurfsansicht. Fügen Sie nun wie in Abbildung 4.1 zwei Kombinationsfelder hinzu, deren Beschriftungen *Bankverbindung* und *Konto/Depot* lauten und die *cboContacts* und *cboAccounts* heißen (wir weichen hier ausnahmsweise von der sonst in diesem Buch üblichen deutschen Benennung von Steuerelementen und Variablen ab, um die Benennung der verwendeten DDBAC-Komponente weiterzuverwenden und nicht allzu vielDurcheinander zu erzeugen).

Eine Datenherkunft für das Formular oder Datensatzherkünfte für die Kombinationsfelder in Form einer Tabelle oder Abfrage brauchen wir vorerst nicht. Die Kombinationsfelder füllen wir dynamisch mit VBA-Funktionen, welche die mit dem DDBAC-Assistenten erstellten Bankverbindungen und Konten auslesen.

-8	frmOnlinebanking	-		×
	1 2 3 4 5 6 7 8 9 10	· 11 ·	· 12 ·	1 🔺
	✓ Formularkopf			
- - 1	Bankverbindungen und Konten/Depots			
<u>-</u>	Bankverbindung: Ungebunden	•		
1	Konto/Depot: Ungebunden	-		

Abbildung 4.1: Das Formular zur Auswahl der Bankverbindung im Entwurf

Kapitel 4 Konten per VBA verwalten

Beim ersten Öffnen des Formulars soll dieses automatisch die Datensatzherkunft des Kombinationsfeldes *cboContacts* füllen. Nach der Auswahl eines Eintrags durch Benutzer trägt eine entsprechende Prozedur dann die Kontos/Depots dieser Bankverbindung in die Datensatzherkunft des zweiten Kombinationsfelds *cboAccounts* ein.

### 4.2 Bankverbindungen einlesen

Als Erstes lesen wir die Bankverbindungen ein und schreiben die resultierende Liste als Datensatzherkunft in die entsprechende Eigenschaft des Kombinationsfeldes *cboContacts*. Dies erledigen wir innerhalb der Ereignisprozedur, die durch die Ereigniseigenschaft *Beim Laden* des Formulars ausgelöst wird. Diese sieht zunächst ganz einfach so aus:

```
Private Sub Form_Open(Cancel As Integer)
   Me!cboContacts.RowSource = GetContactList
End Sub
```

Nun fehlt natürlich noch die Funktion GetContactList, die wir selbst anlegen müssen.

Für diese und alle weiteren für den Zugriff auf die DDBAC-Komponenten nötigen Prozeduren legen wir ein eigenes Standardmodul namens *mdlOnlinebanking* an. Um die *DDBAC*-Objekte zu nutzen, setzen Sie über den *Verweise*-Dialog (VBA-Editor, Menüeintrag *Extras*/*Verweise*) zunächst einen Verweis auf die Bibliothek *DataDesign DDBAC HBCI Banking Application Components*, die Sie durch die oben beschriebene Installation zum System hinzugefügt haben (siehe Abbildung 4.2).

Verweise - prjOnlinebanking	<b>×</b>
Verfügbare Verweise:	ОК
Visual Basic For Applications     Microsoft Access 15.0 Object Library     OLE Automation	Abbrechen
Mind usoft Office 15.0 Access database engine object     DataDesign DDBAC HBCI Banking Application Compor     Heresoft Web, v6.0     Acrobat WebCapture 1.0 Type Library	
Acrobat WebCapture IE Toolbar/Favorites 1.0 Type I     AcroBrokerLib     AcroIEHelper 1.0 Type Library     AcroIEHelperShim 1.0 Type Library     AcroIEHelperShim 1.0 Type Library	Hilfe
Active DS Type Library ActiveMovie control type library	
DataDesign DDBAC IBAN Konverter	
Pfad: \\vmware-host\Shared Folders\Dokumente\Daten\ Sprache: Voreinstellung	Buecher \Onlineb

Abbildung 4.2: Verweis auf die DDBAC-Bibliothek

#### Bankverbindungen einlesen

Für den Zugriff auf die Bibliothek benötigen wir zunächst eine Objektvariable, die einen Verweis auf ein Objekt des Typs *BACContacts* erhalten soll. Dieses macht die einzelnen *BACContact*-Objekte verfügbar. Ein *BACContact*-Objekt entspricht einem Kontakt, den Sie mit dem oben beschriebenen DDBAC-Assistenten angelegt haben.

Da Sie normalerweise mehrmals während einer Session auf die *BACContacts*-Auflistung zugreifen werden, gestalten wir den Zugriff darauf so, dass Sie dieses beim ersten Zugriff erzeugen und bei jedem weiteren Zugriff auf diese Instanz zugreifen können – bis diese beabsichtigt oder unbeabsichtigt (etwa durch einen unbehandelten Laufzeitfehler) gelöscht wird. In diesem Fall erhalten Sie einfach einen Verweis auf eine neue Instanz.

Im Modul *mdlOnlinebanking* deklarieren Sie zunächst eine private Variable namens *m\_Contacts*, welche gleich mit dem Objektverweis gefüllt wird:

Private m\_Contacts As BACContacts

Nun können Sie natürlich nicht von überall auf diese private deklarierte Variable zugreifen, weshalb wir nun eine Funktion zum Modul hinzufügen, die den Zugriff auf den in der Variable gespeicherten Verweis ermöglicht.

Diese Funktion prüft, ob *m\_Contacts* bereits einmal gefüllt wurde, und liefert dann den bestehenden Objektverweis zurück oder sie erstellt einfach ein neues *BACContacts*-Objekt und reicht dieses dann weiter. Vorher füllt die Routine die Auflistung noch mit der *Populate*-Methode (leere Zeichenkette als Parameter nicht vergessen!), damit jederzeit der aktuelle Stand zurückgeliefert wird (siehe Modul *mdlOnlinebanking\_Objekte*):

```
Public Function GetContacts() As BACContacts
    If m_Contacts Is Nothing Then
        Set m_Contacts = New BACContacts
    End If
    m_Contacts.Populate ""
    Set GetContacts = m_Contacts
End Function
```

Der Clou hierbei ist, dass Sie somit keine globale Variable haben, die versehentlich gelöscht werden kann.

Die folgende Funktion namens *GetContactList* schließlich liefert die Zeichenkette, die Sie als Datensatzherkunft des Kombinationsfelds *cboContact* verwenden können. Damit dieses eine Zeichenkette als Datensatzherkunft akzeptiert, müssen Sie noch die Eigenschaft *Herkunftsart* auf den Wert *Wertliste* einstellen.

Alternativ können Sie auch die folgende Zeile als erste Anweisung in die Prozedur *Form\_Load* des Formulars *frmOnlinebanking* integrieren (das gleiche sollten Sie dann auch für das Kombinationsfeld *cboAccounts* erledigen):

Kapitel 4 Konten per VBA verwalten

```
Me!cboContacts.RowSourceType = "Value List"
```

Stellen Sie gegebenenfalls auch noch die Eigenschaft *Wertlistenbearbeitung zulassen* auf *Nein* ein. Die Funktion *GetContactList* liefert eine Zeichenkette wie die folgende zurück – hier mit nur einem Bank-Kontakt:

```
0;1234567890;12121212;Commerzbank |12121212|1234567890;
```

Die Funktion GetContactList sieht schließlich wie folgt aus:

```
Public Function GetContactList() As String
  Dim objContact As BACContact
  Dim strContacts As String
  Dim i As Integer
  For Each objContact In GetContacts
    strContacts = strContacts & i & ":" & objContact.UserID & ":" _
        & objContact.BankCode & ":" _
        & objContact("Contact") _
        & "|" & objContact.UserID & ":"
        i = i + 1
    Next objContact
    End Function
```

Die Prozedur deklariert zunächst ein Objekt namens *objContact*, das einen Bank-Kontakt repräsentiert. Die Zeichenkette *strContacts* soll den späteren Rückgabewert zwischenspeichern, und *i* ist eine Zählervariable.

Danach durchläuft die Prozedur die Elemente der mit der Funktion *GetContacts* gelieferten Auflistung und schreibt diese in eine Zeichenkette, die zuerst den Wert der Zählervariable *i*, dann ein Semikolon, einmal das Benutzerkonto allein und schließlich die drei Informationen *Bankname*, *Bankleitzahl* und *Benutzerkonto* aneinanderhängt. Die hinteren drei Informationen werden dabei durch das Pipe-Zeichen (|) voneinander getrennt. Wenn mehr als ein Kontakt vorliegt, werden die übrigen unter Angabe einer jeweils um eins erhöhten Zählervariablen an die Zeichenkette angehängt. Die Kundennummer, die an zweiter Position der Semikola-separierten Liste gespeichert wird, brauchen wir erst später.

Die letzte Anweisung schreibt das Ergebnis schließlich in den Rückgabewert der Funktion, damit dieser auch der Eigenschaft *RowSource* des Kombinationsfeldes zugewiesen werden kann.

Damit das Kombinationsfeld jeweils den Wert der Zählervariablen (also den Wert vor dem Semikolon) als gebundenen, nicht sichtbaren Wert, das Benutzerkonto allein ebenfalls unsichtbar und die durch die Pipe-Zeichen voneinander getrennten Informationen als sichtbaren Wert anzeigt, stellen Sie noch die Eigenschaften *Spaltenanzahl* und *Spaltenbreiten* auf die Werte 4 und *Ocm;Ocm;Ocm* ein (gleiches gilt für das Kombinationsfeld *cboAccounts*). Die dritte Spalte wird so über die gesamte Breite des Kombinationsfelds angezeigt, die dritte erscheint wie die erste gar nicht.

Wenn Sie das Formular nun in der Formularansicht öffnen und das Kombinationsfeld aufklappen, finden Sie etwa die Ansicht aus Abbildung 4.3 vor.

-8	frmOnlinbanking		5	۵	23
	Bankverbindung:		-	•	
	Konto/Depot:	Commerzbank   35040038   638415109800 Postbank privat   76010085   937289858			
		DataDesign Demobank FinTS3 70000997 111130	15	2	
				,	
Da	itensatz: 🛛 🚽 🕇 von :	1 → N → S Kein Filter Suchen			

Abbildung 4.3: Auswahl des bislang einzigen Bank-Kontakts per Kombinationsfeld

Nun wollen wir zunächst dafür sorgen, dass der Inhalt des zweiten Kombinationsfeldes in Abhängigkeit von dem im ersten Kombinationsfeld ausgewählten Bank-Kontakt gefüllt wird. Dazu kehren wir zunächst wieder zum Modul *mdlOnlinebanking* zurück. Wir brauchen eine Funktion, die in Abhängigkeit vom ausgewählten Kontakt die entsprechenden Konten und Depots zurückliefert. Diese arbeitet prinzipiell genauso wie die Funktion *GetContacts* und sieht wie folgt aus:

```
Public Function GetAccountList(intContact As Integer) As String
   Dim objContact As BACContact
   Dim objAccount As BACAccount
   Dim strAccounts As String
   Dim i As Integer
   Set objContact = GetContacts.Item(intContact)
   For Each objAccount In objContact.Accounts
       strAccounts = strAccounts & i & ";"
            & objAccount.AccountNumber & ";"
           & objAccount.BIC & ";" & objAccount.IBAN & ";"
           & objAccount.AccountNumber _
           & "|" & objAccount.AcctName & ";"
        j = j + 1
   Next objAccount
   GetAccountList = strAccounts
End Function
```

Kapitel 4 Konten per VBA verwalten

Die Funktion erwartet allerdings einen Parameter, und zwar den Index des zu durchsuchenden Kontakts. Dieser ist nullbasiert, entspricht also genau den Werten der ersten Spalte der Datensatzherkunft des Kombinationsfeldes *cboContacts* (welch ein Zufall!) und wird gleich der *Item*-Eigenschaft der mit der Funktion *GetContacts* gelieferten Auflistung als Parameter übergeben. Dies wird, wie bereits in der Funktion *GetContacts*, ständig aktuell über die Funktion *GetContacts* bezogen.

Die Objektvariable *objContact* nimmt schließlich den per Parameterwert identifizierten Eintrag der von *GetContacts* gelieferten Auflistung auf. Elemente vom Typ *BACContact* enthalten wiederum eine Auflistung namens *Accounts*, die wir im Folgenden durchlaufen und dabei einige Informationen zu einer Zeichenkette zusammenstellen. Diese sieht beispielsweise so aus (in einer Zeile):

```
0:411890701:COBADEFFXXX:DE38350400380411890701:411890701|Kontokorrent:
1:415109801:COBADEFFXXX:DE24350400380415109801:415109801|Kontokorrent:
2:415109891:::415109891|Spar:
3:415109801:::415109801|Depot:
```

Diese Zeichenkette soll im Kombinationsfeld *cboAccounts* des Formulars *frmOnlinebanking* vier Einträge ergeben, wobei die Indexwerte 0, 1, 2 und 3 wieder in der unsichtbaren erste Spalte landen – genau wie die Kontonummer aus der zweiten Spalte, der BIC aus der dritten und die IBAN aus der vierten Spalte. Erst die fünfte Spalte mit einem Wert wie 411890701/Kontokorrent soll angezeigt werden. Dazu stellen Sie zunächst die Eigenschaft *Herkunftstyp* auf *Wertliste* und die Eigenschaften *Spaltenanzahl* und Spaltenbreiten auf die Werte 5 und *Ocm;Ocm;Ocm* ein.

Schließlich soll *cboAccounts* nach der Auswahl eines Eintrags in *cboContacts* aktualisiert werden. Dazu legen Sie für das Ereignis *Nach Aktualisierung* die folgende Ereignisprozedur an:

```
Private Sub cboContacts_AfterUpdate()
    If Not IsNull(Me!cboContacts) Then
        Me!cboAccounts.RowSource = GetAccountList(Me!cboContacts)
        Me!cboAccounts = Me!cboAccounts.ItemData(0)
        Me!cboAccounts.SetFocus
        Me!cboAccounts.Dropdown
    Else
        Me!cboAccounts.RowSource = ""
    End If
End Sub
```

Da hier möglicherweise bereits ein Wert für einen anderen Kontakt ausgewählt wurde, soll auch noch der erste Eintrag des Kombinationsfeldes *cboAccounts* ausgewählt und das Kombinationsfeld nach Fokuserhalt aufgeklappt werden. Das Ergebnis sieht dann wie in Abbildung 4.4 aus.

Leseprobe! Vollständiges Buch erhältlich unter www.amvshop.de im André Minhorst Verlag.

#### Bankverbindungen einlesen

-8	frmOnlinebanking	(	-		23
⊁	Bankverbindung:	Commerzbank   150-01101   151-01100   1		[	•
	Konto/Depot:	Kontokorrent		[	-
		Kontokorrent			
		+ 1 ( Kontokorrent			
		+ 1 1 (Spar			
		Depot			
				5	
Da	tensatz: 🛛 斗 1 von 1	L > N > Kein Filter Suchen			

Abbildung 4.4: Nach der Auswahl eines Bank-Kontakts offeriert das Formular auch dessen Konten und Depots.

Da wir gegebenenfalls noch einmal eine Auflistung der *BACAccount*-Objekte benötigen werden, haben wir ihre Ermittlung, die ja auf der Übergabe des Kontakt-Index basiert, in eine eigene Funktion ausgelagert:

```
Public Function GetAccounts(intContact As Integer) As BACAccounts
  Dim objContact As BACContact
  Set objContact = GetContacts.Item(intContact)
  Set GetAccounts = objContact.Accounts
End Function
```

Die obige Funktion *GetAccountList* müssen Sie dann so anpassen, um auf diese Routine zuzugreifen. Außerdem haben wir noch eine Prüfung und einen weiteren Parameter namens *bolSepa* untergebracht. Mit *bolSepa* geben Sie beim Aufruf an, ob nur Sepa-fähige Accounts zurückgegeben werden sollen. Die *If...Then*-Bedingung innerhalb der Schleife prüft nun, ob entweder *bolSepa* den Wert *False* hat, was dazu führt, dass einfach alle Accounts zurückgegeben werden. Oder *bolSepa* hat den Wert *True*: Dann prüft die Funktion zusätzlich für jeden Account, ob dessen Eigenschaft *IsSepa* den Wert *True* hat und gibt diesen nur in diese Fall zurück:

Kapitel 4 Konten per VBA verwalten

## 4.3 Letzten Contact und Account speichern

Ein kleines Feature des Formulars soll sein, dass der zuletzt verwendete Bank-Kontakt und das letzte Konto oder Depot gespeichert werden, damit das Formular diese beim nächsten Öffnen gleich wieder herstellen kann.

Dazu sollen die relevanten Informationen automatisch in der bereits vorhandenen Optionstabelle gespeichert werden, damit der Benutzer beim erneuten Öffnen gleich auf die Bankverbindung und die Kontodaten zugreifen kann, die beim letzten Zugriff verwendet wurden.

Dazu fügen Sie zur Tabelle *tblOptionen* zwei weitere Felder hinzu, und zwar *LetzterBankkontakt* und *LetztesKonto* (siehe Abbildung 4.5).

🛄 tblOptionen		ł
Z Feldname	Felddatentyp	1
LetzterBankkontakt	Zahl	
LetztesKonto	Zahl	- 1

Abbildung 4.5: Erweiterung der Tabelle tblOptionen zum Speichern der zuletzt verwendeten Bankdaten

Nun brauchen wir eine Funktion, die dafür sorgt, dass die verwendete Konfiguration vor dem Schließen des Formulars in der Tabelle *tblOptionen* gesichert wird. Das Schließen selbst übernimmt die Schaltfläche *cmdOK*, welche die folgende Ereignisprozedur auslöst und somit zunächst lediglich das Formular schließt:

```
Private Sub cmdOK_Click()
    DoCmd.Close acForm, Me.Name
End Sub
```

Dort bringen wir den erforderlichen Code nicht unter, denn das Formular kann ja auch auf anderen Wegen geschlossen werden – etwa über die Schließen-Schaltfläche oben rechts im Formular oder auch durch Beenden der Anwendung. Das passende Ereignis für diesen Fall heißt Beim Entladen. Dafür hinterlegen wir die folgende Ereignisprozedur:

#### Letzten Contact und Account speichern

Nachdem wir das Formular ein wenig hübscher gemacht haben, sieht dieses wie in Abbildung 4.6 aus. Dazu haben wir die Eigenschaften *Navigationsschaltflächen, Bildlaufleisten, Trennlinien* und *Datensatzmarkierer* auf *Nein* und *Automatisch zentrieren* auf *Ja* eingestellt. Außerdem enthält der Formularkopf nun ein Bildsteuerelement mit einem hübschen Icon und eine entsprechende Überschrift.



Abbildung 4.6: Auswahl von Bank-Kontakt und Konto/Depot

Ein Klick auf *OK* speichert nun die ausgewählten Einstellungen, die wie in Abbildung 4.7 in der Tabelle *tblOptionen* landen.

E	tblOptioner	ı	-		×
1	LetzterBankkontakt	-	LetztesKonto	-	
		1		1	
*					
Di	atensatz: I4 → 1 von 1 →		🕨 🛤 🛛 🏹 Kein F	ilter	Su

Abbildung 4.7: Die zuletzt im Formular *frmOnlinebanking* gewählte Konfiguration wird hier gespeichert.

Nun fehlt noch eine Erweiterung der Ereignisprozedur *Beim Laden* des Formulars *frmOnlinebanking*, welche die gespeicherten Optionen ausliest und die Kombinationsfelder entsprechend einstellt. Dazu erweitern Sie die Prozedur *Form\_Load* wie folgt um die kursiv gedruckten Zeilen:

```
Private Sub Form_Load()
Dim intLastContact As Integer
```

#### Kapitel 4 Konten per VBA verwalten

```
Dim intLastAccount As Integer
Me!cboContacts.RowSource = GetContacts
intLastContact = Nz(DLookup("LetzterBankkontakt", "tbl0ptionen"), -1)
If Not intLastContact = -1 Then
Me!cboContacts = Me!cboContacts.ItemData(intLastContact)
Me!cboAccounts.RowSource = GetAccounts(intLastContact)
intLastAccount = Nz(DLookup("LetztesKonto", "tbl0ptionen"), -1)
If Not intLastAccount = -1 Then
Me!cboAccounts = Me!cboAccounts.ItemData(intLastAccount)
End If
End If
End Sub
```

Die Routine deklariert zunächst zwei Integer-Variablen zum Zwischenspeichern der in der Tabelle *tblOptionen* enthaltenen Werte. Die erste *DLookup*-Anweisung liest den Wert des Feldes *Letz-terBankkontakt* ein. Ist dieser *Null*, ändert die umschließende *Nz*-Funktion den Rückgabewert auf -1 und schreibt ihn dann in die Variable *intLastContact*. Dies wertet die folgende *If...Then*-Bedingung aus: Ist der Wert -1, bricht sie ab, sonst geht sie davon aus, dass der Kontakt mit dem gespeicherten Index angezeigt werden soll, und erledigt dies auch.

Hier kommt die *ItemData*-Eigenschaft zum Zuge, die den Kombinationsfeldeintrag mit dem angegebenen Index anzeigt (hier *intLastContact*). Da nun ein Eintrag für das erste Kombinationsfeld ausgewählt wurde, kümmert sich die Prozedur zumindest noch darum, dass die Datensatzherkunft des zweiten Kombinationsfelds so eingestellt wird, dass es alle Accounts zu dem im ersten Kombinationsfeld ausgewählten *Contact* enthält. Nachdem dies geschehen ist, fragt eine zweite *DLookup*-Funktion ab, ob in der Tabelle *tblOptionen* auch ein Index für den zuletzt ausgewählten Account vorliegt.

Falls nicht, liefert diese Funktion den Wert *Null* zurück, was wiederum als -1 in der Variablen *intLastAccount* gespeichert wird. In diesem Fall endet die Prozedur hier. Anderenfalls stellt sie auch noch den Wert des zweiten Kombinationsfeldes ein, und zwar auf den durch die *DLookup*-Funktion ermittelten Wert.

Beim Testen der Beispieldatenbanken fiel noch auf, dass es einen Fehler gibt, wenn man dieses und weitere Formulare, die auf die Kontakte zugreifen, öffnet, ohne dass Kontakte angelegt wurden. Also verschieben wir den Inhalt der Prozedur *Form\_Load* in die Prozedur *Form\_Open* und fügen noch eine Prüfung hinzu, um sicherzustellen, dass mindestens ein Kontakt vorhanden ist:

```
Private Sub Form_Open(Cancel As Integer)
Dim intLastContact As Integer
Dim intLastAccount As Integer
Me!cboContacts.RowSourceType = "Value List"
Me!cboAccounts.RowSourceType = "Value List"
Me!cboContacts.RowSource = GetContactList
```

#### Konto-Informationen anzeigen

```
If Len(Me!cboContacts.RowSource) = 0 Then
       MsgBox "Es wurden noch keine Kontakte angelegt." & vbCrLf & vbCrLf
           & "Dies können Sie über den Eintrag 'Homebanking Kontakte' in der " _
           & "Systemsteuerung erledigen."
       Cancel = True
       Exit Sub
   Fnd If
   intLastContact = Nz(DLookup("LetzterBankkontakt", "tbl0ptionen"), -1)
   If Not intLastContact = -1 Then
       Me!cboContacts = Me!cboContacts.ItemData(intLastContact)
       Me!cboAccounts.RowSource = GetAccountList(intLastContact)
       intLastAccount = Nz(DLookup("LetztesKonto", "tbl0ptionen"), -1)
       If Not intLastAccount = -1 Then
           Me!cboAccounts = Me!cboAccounts.ItemData(intLastAccount)
       Fnd If
   End If
End Sub
```

### 4.4 Konto-Informationen anzeigen

Wenn Sie ein Konto synchronisieren, liest die DDBAC einige Informationen über das Konto ein und speichert diese lokal. Diese Informationen stehen über das Objektmodell der DDBAC zur Verfügung. Um diese Informationen einzusehen, fügen Sie dem Formular *frmOnlinebanking* einige Steuerelemente hinzu (siehe Abbildung 4.8).

Kapitel 4 Konten per VBA verwalten

	frmOnlinebanking – 🗖	×
• · · · 1 · · · 2 · · · 3 · · ·	4 • 1 • 5 • 1 • 6 • 1 • 7 • 1 • 8 • 1 • 9 • 1 • 10 • 1 • 11 • 1 • 12 •	
Formularkopf		
Bankverb	indungen und Depots	
Oetailbereich		
Bankverbindung: Unge	ebunden 💌	
1 Konto/Depot: Unge	ebunden 💌	
Informationen		
AccountNumber:	Ungebunden	
3 AcctName:	Ungebunden	
AvailableTransactions:	Ungebunden	
BankCode:	Ungebunden	
5 BIC:	Ungebunden	
CountryCode:	Ungebunden	
- Currency:	Ungebunden	
7. CustomerID:	Ungebunden	Ш
BAN:	Ungebunden	
- IsSepa:	V	
NameEins:	Ungebunden	
10 NameZwei:	Ungebunden	
SubAccountNumber:	Ungebunden	
Formularfuß	·	
Е Кок		

Abbildung 4.8: Anlegen einiger Steuerelemente für die Detailinformationen des ausgewählten Kontos

Nun benötigen Sie eine Prozedur, welche die Steuerelemente nach der Auswahl eines Eintrags im Kombinationsfeld *cboAccounts* aktualisiert. Diese ermittelt zunächst das betroffene *BACContact*-Objekt für den gewählten Eintrag im ersten Kombinationsfeld und speichert einen Verweis darauf in der Variablen *objContact*. Die *Accounts*-Auflistung dieses Objekt erlaubt schließlich das Referenzieren des Kontos selbst. Der Verweis darauf landet in der Variablen *objAccount*. Über dieses Objekt greift die Prozedur auf die folgenden Eigenschaften zu:

- » AccountNumber: Kontonummer
- » AcctName: Bezeichnung des Kontos (zum Beispiel Kontokorrent, Spar, Depot,
- » BankCode: Bankleitzahl
- » BIC: BIC für dieses Konto
- » CountryCode: Länderkennzeichnen für dieses Konto, zum Beispiel 280 für Deutschland

#### Konto-Informationen anzeigen

- » Currency: Währung, zum Beispiel EUR
- » *CustomerID*: Kundennummer, entspricht gelegentlich der Kontonummer. Bei der Commerzbank etwa sind hier noch die Ziffern 638 vorangestellt.
- » IBAN: IBAN für dieses Konto
- » IsSepa: Gibt an, ob es sich um ein SEPA-fähiges Konto handelt
- » NameEins: Name des Kontoinhabers
- » NameZwei: Name des Kontoinhabers
- » SubAccountNumber: vom Kreditinstitut vergebene Bezeichnung

#### Der Code der Prozedur sieht wie folgt aus:

```
Private Sub cboAccounts AfterUpdate()
   Dim objContact As BACContact
   Dim objAccount As BACAccount
   Set objContact = GetContacts.Item(Me!cboContacts)
   Set objAccount = objContact.Accounts(Me!cboAccounts)
   With objAccount
       Me!txtAccountNumber = .AccountNumber
       Me!txtAcctName = .AcctName
       Me!txtBankCode = .BankCode
       Me!txtBIC = .BIC
       Me!txtCountyCode = .CountryCode
       Me!txtCurrency = .Currency
       Me!txtCustomerID = .CustomerID
       Me!txtIBAN = .IBAN
       Me!chkIsSepa = .IsSepa
       Me!txtNameEins = .NameEins
       Me!txtNameZwei = .NameZwei
       Me!txtSubAccountNumber = .SubAccountNumber
   End With
Fnd Sub
```

Nach dem Auswählen des Kontos sieht das Formular in der Formularansicht beispielsweise wie in Abbildung 4.9 aus.

Kapitel 4 Konten per VBA verwalten

	frmOnlinebanking – 🗆 🗙	_
Bankverbi	ndungen und epots	
Bankverbindung: Postb	ank	
Konto/Depot:	Postbank Giro plus	
Informationen		
AccountNumber:	9572#H#58	
AcctName:	Postbank Giro plus	
AvailableTransactions:		
BankCode:	Treat Linkien.	
BIC:	PBNKDEFF	
CountryCode:	280	
Currency:	EUR	
CustomerID:	857289858	
IBAN:	DE57760100850937289858	
IsSepa:		
NameEins:	Andre Minhorst	
NameZwei:		
SubAccountNumber:		
🔀 ок		

Abbildung 4.9: Anzeige der Konto-Informationen des mit den beiden Kombinationsfeldern gewählten Kontos

### 4.5 Dialog zum Verwalten der Konten per VBA öffnen

Den oben genannten Dialog können Sie auch per VBA öffnen. Dazu instanzieren Sie einfach ein Objekt des Typs *BACBanking* und rufen dann die Methode *RunContactsCPL* auf (siehe Modul *mdlOnlinebanking*):

```
Public Sub KontakteVerwalten()
   Dim objBanking As BACBanking
   Set objBanking = New BACBanking
   With objBanking
        .RunContactsCPL
   End With
End Sub
```

Dies zeigt den Dialog in der Standardansicht an.

# **5** Kontostand ermitteln

Dieses Kapitel zeigt, wie Sie die den Kontostand für ein zuvor ausgewähltes Konto ermitteln und verarbeiten. Im vorherigen Kapitel haben Sie erfahren, wie Sie die Daten der Bankverbindung und des gewünschten Kontos ermitteln. In diesem Kapitel schauen wir uns an, wie Sie für eines dieser Konten den aktuellen Kontostand ermitteln. Der Kontostand kommt in Form von vier wichtigen Informationen:

- » Datum des Kontostands
- » Währung
- » Saldo
- » Kennzeichen für Soll/Haben

# 5.1 Betroffenes Konto ermitteln

Das Formular *frmKontostand* soll den Kontostand wie in Abbildung 5.1 darstellen. Die beiden Kombinationsfelder *cboContacts* und *cboAccounts* kennen Sie bereits vom zuvor beschriebenen Formular *frmOnlinebanking*. Sie dienen wiederum der Auswahl des Kontaktes und des Kontos, für welches der Kontostand ermittelt werden soll.

Kapitel 5 Kontostand ermitteln

Ξ	3	frmKontostand	-		×
	· · · 1 · · · 2 · · · :	3 · 1 · 4 · 1 · 5 · 1 · 6 · 1 · 7 · 1 · 8 · 1 · 9 · 1	· 10 · I	• 11 •	-
	Formularkopf				
- - 1	Konto	stand ermitteln			
-	Bankverbindung:	Ungebunden		•	
1	Konto/Depot:	Ungebunden		•	
-	Aktualisieren				
-	Kontostand				
3	Datum:	Ungebunden			
-	Saldo:	Ungebunden			
-	Soll/Haben:	Ungebunden			
5	Währung:	Ungebunden			
-					
: -	К				
1					

Abbildung 5.1: Das Formular frmKontostand mit den vier Steuerelementen zur Anzeige des aktuellen Saldo

Das Einlesen der Informationen über den Kontostand erfolgt über einen Klick auf die Schaltfläche *cmdKontostandAktualisieren*, welche die Funktion *Kontostand* aus dem Modul *mdlOnlinebanking* aufruft und eine Variable des Typs *tKontostand* zurückerhält.

*tKontostand* ist ein benutzerdefinierter Typ, mit dem Sie mehrere Variable auch unterschiedlichen Typs zusammenfassen können. In diesem Fall soll er alle zuvor beschriebenen Elemente enthalten. Seine Deklaration landet im Modul *mdlOnlinebanking* und sieht so aus:

```
Public Type tKontostand
Wert As Currency
Datum As Date
Waehrung As String
SollHaben As String
End Type
```

Die Ereignisprozedur, die durch das Ereignis *Beim Klicken* der Schaltfläche *cmdKontostandAktualisieren* ausgelöst wird, deklariert eine Variable dieses benutzerdefinierten Typs und weist dieser den Rückgabewert der Funktion *Kontostand* zu. Was diese erledigt, erfahren Sie weiter unten.

Im Anschluss sehen Sie, wie Sie mit einem gefüllten benutzerdefinierten Typ umgehen können: Sie können über die Punkt-Syntax auf jedes seiner Elemente zugreifen und diese beispielsweise den vier Textfeldern im Formular *frmKontostand* aus der vorherigen Abbildung zuweisen.

#### **Betroffenes Konto ermitteln**

Wenn der Rückgabewert der Funktion *KontostandErmitteln* den Wert *True* hat, ist *objKonto-stand* mit den benötigten Daten gefüllt. Die Funktion erwartet als Parameter den Index der in den Kombinationsfeldern *cboContacts* und *cboAccounts* ausgewählten *Contact-* und *Account-*Elemente sowie das zu füllende Objekt *objAktuellerKontostand*.

Da die Prozedur ein paar Zeilen länger ist, gliedern wir sie in den Fließtext ein – für die zusammenhängende Übersicht empfehlen wir einen Blick in das Modul *mdlOnlinebanking* der Beispieldatenbank.

#### Ablauf beim Abfragen des Kontostands

Vorab finden Sie einen kurzen Überblick darüber, was die Funktion erledigt:

- » Abfragen der Passphrase (etwa bei Verwendung einer Schlüsseldatei) oder des PIN (bei PIN/ TAN-Verfahren)
- » Aufbauen der Verbindung zum Server der Bank
- » Aufbauen des Segments mit Informationen zu den abzufragenden Daten (etwa Kreditinstitut-Code, Kontonummer ...)
- » Abschicken des Segments und Abholen des Antwort-Segments
- » Auswerten des Antwort-Segments

#### Kontostandabfrage im Detail

Die Funktion deklariert zunächst einige Variablen, deren Bedeutung wir im Verlauf der Prozedur erläutern:

Kapitel 5 Kontostand ermitteln

Dim objAccountSegment As BACSegment Dim objDialog As BACDialog Dim objMessage As BACMessage Dim objSegmentAntwort As BACSegment Dim objSegmentVersion As BACSegment Dim objSegmentVersion As BACSegment Dim objAccount As BACAccount Dim lngMaxSegmentVersion As Long Dim tKontostandTemp As tKontostand Dim strSegment As String

Die Objektvariable *objContact* wird mit dem Element der *BACContacts*-Auflistung gefüllt, das dem mit dem Parameter *intContact* übergebenen Index entspricht:

Set objContact = GetContacts.Item(intContact)

Gleiches gilt für die Variable objAccount:

Set objAccount = objContact.Accounts(intAccount)

Anschließend folgt bereits die Anweisung, die den Dialog zum Eingeben der erforderlichen Benutzerdaten abfragt. Dies erledigt die Methode *NewDialogUI* des *BACContact*-Objekts:

Set objDialog = objContact.NewDialogUI

Der Dialog sieht je nach Zugriffsart (PIN/TAN, Chipkarte, Schlüsseldatei) anders aus. Für eine Schlüsseldatei etwa erscheint der Dialog aus Abbildung 5.2. Wenn Sie mit PIN/TAN arbeiten, erscheint etwa ein Dialog wie in Abbildung 5.3.

Leseprobe! Vollständiges Buch erhältlich unter www.amvshop.de im André Minhorst Verlag.

**Betroffenes Konto ermitteln** 



Abbildung 5.2: Hier geben Sie die Passphrase für die Schlüsseldatei ein, mit der die Anfragen an die Bank verschlüsselt werden.

HBCI/FinTS-Kontakt	
PIN (Passwort) Eingabe	
Für die gewünschte Funktion wird eine gültige P Homebanking Kontakt: DataDesign Demobank FinTS3	IN (Passwort) benötigt.
901 mobile TAN	TAN Liste: 1138157702 -
PIN (Passwort):	1 2 3 4 5 6 7 8 9 C 0 +
Zurück     Weiter	Abbrechen

Abbildung 5.3: Alternativ erscheint dieser Dialog, wenn der Zugriff per PIN/TAN abgesichert ist.

Kapitel 5 Kontostand ermitteln

Anschließend erstellt die folgende Zeile ein *BACAccount*-Segment, also ein Objekt, das den Zugriff auf die Informationen des in diesem Zusammenhang verwendeten Accounts über eine spezielle, gleich vorgestellte Syntax erlaubt:

Set objAccountSegment = objAccount.Segment

Da wir später mehrmals darauf zugreifen, speichern wir nun noch die Kennung des Segments in einer *String*-Variablen:

strSegment = "HKSAL"

Damit sind die Vorbereitungen abgeschlossen. Die Routine prüft nun noch, ob *objDialog* einen Objektverweis enthält. Dies ist zum Beispiel dann nicht der Fall, wenn der Dialog zur Eingabe der Passphrase mit der *Abbrechen*-Schaltfläche beendet wurde. Anderenfalls wird nun eine Verbindung zum Server der Bank aufgenommen:

If Not objDialog Is Nothing Then objDialog.BeginDialog

Anschließend ermittelt die Prozedur anhand der Funktion *FindOptimalBPDSegment* die aktuelle Version des für diesen Vorgang, also die Abfrage eines Kontos, benötigten Segments. *FindOptimalBPDSegment* erwartet die Kennung des Segments, hier *HKSAL*, und eine leere *Long*-Variable, mit der Sie die Versionsnummer übergeben können.

Hier übergeben wir den Standardwert *0*, was automatisch dazu führt, dass die neueste Version des Segments verwendet wird:

```
Set objSegmentVersion = objAccount.FindOptimalBPDSegment(strSegment, 7
IngMaxSegmentVersion)
```

Für bestimmte Konten steht das Segment *HKSAL* gar nicht zur Verfügung – etwa für Depots. In diesem Fall bleibt *objSegmentVersion* leer. Dies prüfen wir gleich im Anschluss. Nur wenn *obj-SegmentVersion* nicht *Nothing* entspricht, werden die nachfolgend beschriebenen Anweisungen ausgeführt, wofür folgende *If...Then*-Bedingung sorgt:

```
If Not objSegmentVersion Is Nothing Then
    '... siehe weiter unten
Else
    KontostandErmitteln = False
End If
```

Auf Basis der mit dem Objekt *objSegmentversion* zurückgelieferten Versionsnummer, verfügbar über die Eigenschaft *Version*, erstellt die folgende Anweisung ein neues *BACSegment*-Objekt namens *objSegmentAnfrage* mit dem Segmenttyp *HKSAL*:

#### **Betroffenes Konto ermitteln**

Die folgenden Zeilen bestücken das Segment nun mit den benötigten Informationen. Die Parameter wie *AuftraggeberKontoverbindung1*, *Kontonummer1* et cetera können Sie, wenn Sie eigene Anfragen zusammenstellen möchten, über die im Kasten *Dokumentation der Segmenttypen* angegebenen Dokumentationen finden. Hier wird nun auch klar, wozu das Objekt *objAccount-Segment* nötig ist – es enthält nämlich all die Informationen zu dem Konto, für das wir den Kontostand auslesen wollen.

Die benötigten Elemente erhält man, wenn man die entsprechenden Parameter übergibt:

Schließlich führen wir die Anfrage aus, und zwar mit der *ExecuteSegment*-Methode des Objekts *objDialog*:

Set objMessage = objDialog.ExecuteSegment(objSegmentAnfrage)

Das Antwortsegment finden wir anschließend in der Auflistung *objMessage.Transactions(0). ResponseSegments*. Diese Auflistung kann je nach Anfrage-Segment auch mehrere Elemente enthalten, zum Beispiel beim nachfolgend vorgestellten Segment zum Abfragen der Umsätze.

Nachdem wir das Ergebnis mit dem Objekt *objMessage* erhalten haben, schicken wir es kurz zur Prozedur *TransaktionAnalysieren*, welche die Ergebnisse im XML-Format in die Protokolltabelle schreibt:

TransaktionAnalysieren strSegment, objMessage

Zurück zu den *ResponseSegments*: Hier gibt es nur ein Segment, dessen Inhalt über die folgenden Zeilen ermittelt und – ganz wichtig – in die einzelnen Elemente unseres speziell für diesen Zweck erstellen benutzerdefinierten Typen geschrieben werden:

```
For Each objSegmentAntwort In objMessage.Transactions(0).ResponseSegments
With objSegmentAntwort
tKontostandTemp.Wert = .Item("SaldoGebucht1", "Wert1")
tKontostandTemp.Datum = .Item("SaldoGebucht1", "Datum1")
tKontostandTemp.Waehrung = .Item("SaldoGebucht1", "Waehrung1")
tKontostandTemp.SollHaben =
```

Kapitel 5 Kontostand ermitteln

```
.Item("SaldoGebucht1", "SollHabenKennzeichen1")
End With
Next objSegmentAntwort
```

Schließlich wird der Dialog mit dem Bankserver über die Methode *EndDialog* beendet und der benutzerdefinierte Typ *tKontostandTemp* gefüllt an den Rückgabeparameter *objKontostand* der Funktion übergeben. Der Funktionswert *KontostandErmitteln* wird mit dem Wert *True* gefüllt:

```
objDialog.EndDialog
objKontostand = tKontostandTemp
KontostandErmitteln = True
End If
End Function
```

Und damit kommen wir zurück zum Ausgangspunkt – dem Formular *frmKontostand*, dessen Felder nun mit den Informationen zum aktuellen Kontostand gefüllt werden (siehe Abbildung 5.4).

-8	== frmKontostand					
Kontostand ermitteln						
Bankverbindung:	Postbank		•			
Konto/Depot:	Postbank Giro plus		•			
ୠ Aktualisieren						
Kontostand —	••					
Datum:	10.09.2014					
Saldo:	2527.2					
Soll/Haben:	D					
Währung:	EUR					
🔀 ок						

Abbildung 5.4: Das Formular frmKontostand mit dem aktuellen Kontostand

## 5.2 Alle Konten einlesen

Nun haben Sie vielleicht weiter oben den Parameter *AlleKonten1* erspäht. Was macht dieser? Richtig: Damit geben Sie an, ob alle Konten eines Kontaktes eingelesen werden sollen. Wir wollen uns auch noch anschauen, wie dies gelingt. Interessant ist dies natürlich für eine Finanzübersicht.

Also bauen wir ein weiteres Formular auf, dass diesmal *frmKontostand\_AlleKonten* heißt. Es verwendet wiederum die beiden Kombinationsfelder aus dem zuvor beschriebenen Formular *frmKontostand*. Der Code sieht allerdings an einigen Stellen wesentlich anders aus, weil wir

#### Alle Konten einlesen

nicht nur jeweils einen Kontakt und ein Konto dieses Kontakts auswählen möchten, sondern auch die Möglichkeit anbieten möchten, alle Konten eines Kontaktes abzufragen und sogar auch alle Konten aller Kontakte. Zudem können die empfangenen Daten nun nicht mehr in nur vier Textfeldern untergebracht werden, daher verwenden wir nun ein Unterformular in der Datenblattansicht. Und wo ein solches ist, kann auch eine Tabelle zum Speichern der darin angezeigten Daten nicht weit sein – diese heißt *tblKontostaende* und sieht wie in Abbildung 5.5 aus.

Abbildung 5.5: Tabelle zum Speichern von Kontoständen

Diese speichert den jeweils aktuellen Stand für ein Konto. Die Darstellung erfolgt wie in der Formularansicht aus Abbildung 5.6. Neu hinzugekommen im Vergleich zum vorherigen Formular ist lediglich das Unterformularsteuerelement mit dem Unterformular *sfmKontostaende*. Dieses verwendet die Tabelle *tblKontostaende* als Datenherkunft und zeigt alle Felder dieser Tabelle mit Ausnahme des Primärschlüsselfelds *KontostandID* an.

Kapitel 5 Kontostand ermitteln

-5	FS frmKontostand_AlleKonten							×				
Alle Kontostände ermitteln												
Bankverbindung: <alle anzeigen=""></alle>					•							
Konto/Depot: <a></a> <				•								
	Aktualisieren											
Kor	ntostände:							_				
	BLZ 👻	Teilnehmer 🗸	Konto 🗸	Bezeichnung 👻	Datum 🚽	Saldo 🚽	Waehrung					
	35040038	63841	4151(	Kontokorrent	08.09.2014		EUR					
	35040038	63841	4151(	Spar	31.12.2013		EUR					
	76010085	93728	3415	PB Giro plus	10.09.2014		EUR					
	76010085	93728	9372	PB Giro plus	10.09.2014		EUR					
	76010085	93728	25054	PB Giro plus	10.09.2014		EUR					
	76010085	93728	4184	PB Giro plus	10.09.2014		EUR					
	76010085	93728	3338	PB Tagesgeld	10.09.2014		EUR					
	76010085	93728	10434	PB Tagesgeld	10.09.2014		EUR					
	76010085	93728	49074	VISA Card	10.09.2014		EUR					
	35040038	63841	4118	Kontokorrent	08.09.2014		EUR					
*						0,00 €						
Da	atensatz: 14	1 von 10 → →	Kein Filter	Suchen								
			~									

Abbildung 5.6: Das Formular frmKontostand\_AlleKonten in der Formularansicht

Mit dem Formular können Sie nun die Kontostände eines einzigen Kontos, aller Konten eines Kontaktes oder aller Konten aller Kontakte gleichzeitig abrufen:

- » Für den einfachen Kontoabruf wählen Sie den gewünschten Kontakt und das gewünschte Konto mit den beiden Kombinationsfeldern aus und klicken dann auf die Schaltfläche cmd-KontostandAktualisieren.
- » Wollen Sie alle Konten eines Kontakts abrufen, wählen Sie im oberen Kombinationsfeld den gewünschten Kontakt aus und im unteren den Eintrag *<Alle anzeigen>*.
- » Wenn alle Konten aller Kontakte eingelesen werden sollen, wählen Sie im oberen wie auch im unteren Kombinationsfeld jeweils den Eintrag <*Alle anzeigen>* aus.

Gleichzeitig mit der Auswahl dieser Einträge passt das Formular auch die angezeigten Werte im Unterformular an. Diese werden entsprechend der Auswahl gefiltert, sodass Sie dort alle Konten, nur die Konten eines Kontakts oder auch nur ein einziges Konto anzeigen können.

In der Entwurfsansicht sieht das Formular wie in Abbildung 5.7 aus.

Leseprobe! Vollständiges Buch erhältlich unter www.amvshop.de im André Minhorst Verlag.

Alle Konten einlesen

I frmKontostand_AlleKonten	-		×
• • • • • • • • • • • • • • • • • • • •	16 י י	· 17 ·	1 🔺
✓ Formularkopf			_
Alle Kontostände ermitteln			
Detailbereich			-
Bankverbindung: Ungebunden			
1 Konto/Depot: Ungebunden			
2 Kontostände:			
3         -         -         1         1         -         1         1         -         1         1         -         1	. • 1 •	16 📥	
4 - BLZ: BLZ			
5 1 Teilnehmer: Teilnehmer			
- Konto: Konto			
6 2 Bezeichnung: Bezeichnung	ļ	<ul><li>▼</li><li></li></ul>	
₹ Formularfuß			
- К ок			-

Abbildung 5.7: Die Entwurfsansicht des Formulars frmKontostand\_AlleKonten

### 5.2.1 Öffnen des Formulars

Schauen wir uns also an, wie die Ereignisprozeduren hinter den Steuerelementen aussehen. Als Erstes müssen wir uns jedoch um die Prozedur kümmern, die durch das Ereignis *Beim Öffnen* des Formulars ausgelöst wird. Dieses sieht wie folgt aus:

```
Private Sub Form_Open(Cancel As Integer)
Dim intLastContact As Integer
Dim intLastAccount As Integer
Dim strContacts As String
Dim strAccounts As String
Me!cboContacts.RowSourceType = "Value List"
Me!cboAccounts.RowSourceType = "Value List"
strContacts = GetContactList
strContacts = GetContactList
Me!cboContacts.RowSource = strContacts
If Len(Me!cboContacts.RowSource) = 0 Then
MsgBox "Es wurden noch keine Kontakte angelegt." & vbCrLf & vbCrLf
& "Dies können Sie über den Eintrag 'Homebanking Kontakte' in der "
& "Systemsteuerung erledigen."
```

#### Kapitel 5 Kontostand ermitteln

```
Cancel = True
       Exit Sub
   Fnd If
   intLastContact = Nz(DLookup("LetzterBankkontakt", "tbl0ptionen"), -1)
   Me!cboContacts = intLastContact
   If Not intlastContact = -1 Then
       strAccounts = GetAccountList(intLastContact)
       strAccounts = "-1;;;;<Alle anzeigen>;" & strAccounts
       Me!cboAccounts.RowSource = strAccounts
       intLastAccount = Nz(DLookup("LetztesKonto", "tbl0ptionen"), -1)
       Me!cboAccounts = intLastAccount
   Flse
       Me!cboAccounts.RowSource = "-1;;;;;<Alle anzeigen>;"
       Me!cboAccounts = Me!cboAccounts.ItemData(0)
   Fnd If
   UnterformularAktualisieren
End Sub
```

Die Prozedur stellt auch wieder die Herkunftsart der beiden Kombinationsfelder auf Wertliste ein. Dann ermittelt sie, wie zuvor, die Liste der Kontakte über die Funktion *GetContactList*. Das Ergebnis landet diesmal allerdings nicht direkt in der Eigenschaft *Datensatzherkunft* des Kombinationsfeldes, sondern in der String-Variablen *strContacts*. Wir müssen den Daten nämlich noch den Eintrag *<Alle auswählen>* hinzufügen, was die folgende Anweisung erledigt. Da das Kombinationsfeld fünf Spalten enthält, müssen wir neben dem Wert für die gebundene Spalte und dem anzuzeigenden Wert noch ein paar leere Werte vorsehen – etwa so: *-1;;;<Alle anzeigen>;*. Danach fügen wir dem Kombinationsfeld die Datensatzherkunft hinzu.

Ist die Datensatzherkunft leer, wird das Öffnen mit einer Meldung abgebrochen – offensichtlich sind in diesem Fall noch keine Kontakte über die Systemsteuerung angelegt worden. Anderenfalls ermittelt die Prozedur den zuletzt verwendeten Kontakt aus der Tabelle *tblOptionen* und stellt das Kombinationsfeld auf diesen Eintrag ein. Hat dieser nicht den Wert -1, bedeutet dies, dass ein entsprechender Wert in der Tabelle *tblOptionen* gefunden und ein Eintrag im ersten Kombinationsfeld ausgewählt wurde. In diesem Fall ermittelt die Prozedur auch noch die Einträge für das zweite Kombinationsfeld. Auch hier fügt sie wieder den Eintrag *<Alle anzeigen>* vorne an und stellt das Kombinationsfeld dann auf den zuletzt verwendeten Wert ein.

Sollte zuvor kein Kontakt ausgewählt und in der Tabelle *tblOptionen* gespeichert worden sein, stellt die Prozedur das zweite Kombinationsfeld *cboAccounts* auf eine Datensatzherkunft ein, die lediglich den Eintrag *<Alle anzeigen>* enthält und wählt diesen gleich als aktuelle Eintrag aus.

Im Anschluss aktualisiert die Prozedur durch einen Aufruf der Prozedur UnterformularAktualisieren noch das Unterformular. Leseprobe! Vollständiges Buch erhältlich unter www.amvshop.de im André Minhorst Verlag.

Alle Konten einlesen

### 5.2.2 Unterformular aktualisieren

Diese Prozedur UnterformularAktualisieren wird nicht nur beim Öffnen des Formulars aufgerufen, sondern auch noch beim Auswählen von Einträgen mit den beiden Kombinationsfeldern *cboContacts* und *cboAccounts*. Sie sieht wie folgt aus:

```
Private Sub UnterformularAktualisieren()
   Dim strKonto As String
   Dim strTeilnehmer As String
   Dim strBLZ As String
   Dim strFilter As String
   If Not Me!cboContacts = -1 Then
       strBLZ = GetContacts.Item(Me!cboContacts).BankCode
       strTeilnehmer = GetContacts.Item(Me!cboContacts).UserID
       strFilter = "BLZ = '" & strBLZ & "' AND Teilnehmer = '" & strTeilnehmer & "'"
       If Not Me!cboAccounts = -1 Then
            strKonto = GetAccounts(Me!cboContacts).Item(Me!cboAccounts).AccountNumber
           strFilter = strFilter & " AND Konto = '" & strKonto & "'"
       End If
   End If
   Me!sfmKontostaende.Form.Filter = strFilter
   Me!sfmKontostaende.Form.FilterOn = True
Fnd Sub
```

Die Prozedur prüft zunächst, ob das erste Kombinationsfeld *cboContacts* nicht den Wert -1 enthält. Dies würde bedeuten, dass der Benutzer einen anderen Eintrag als *<Alle anzeigen>* ausgewählt hat. Hat der Benutzer diesen Wert gewählt, würde der Inhalt der *If…Then*-Bedingung nicht ausgeführt und somit bliebe auch die Variable *strFilter* leer. Dies würde weiter dazu führen, dass die Eigenschaft *Filter* des Unterformulars mit einer leeren Zeichenkette gefüllt wird, was zur Anzeige aller vorhandenen Datensätze führt.

Doch schauen wir uns den Fall an, dass der Benutzer einen bestimmten Kontakt ausgewählt hat. Dann füllt die Prozedur die Variable *strBLZ* mit der Eigenschaft *BankCode* des entsprechenden *BACContact*-Elements. Dieses beziehen wir über die Funktion *GetContacts*, die eine Auflistung aller *BACContact*-Elemente liefert. Außerdem füllt die Prozedur die Variable *strTeilnehmer* mit der Eigenschaft *UserID* des gleichen *BACContact*-Elements. Dies entspricht der *Teilnehmernummer* für diesen Kontakt. Aus diesen beiden Informationen setzt die Prozedur dann den ersten Teil des Filterausdrucks zusammen, der beispielsweise wie folgt aussieht:

```
BLZ = '35040123' AND Teilnehmer = '638415101234'
```

Anschließend prüft die Prozedur noch, ob auch in *cboAccounts* ein anderer Eintrag als -1, also <*Alle anzeigen>*, ausgewählt wurde. In diesem Fall landet die Eigenschaft *AccountNumber* des

Kapitel 5 Kontostand ermitteln

entsprechenden *BACAccount*-Objekts in der Variablen *strKonto*. Dieses hängt die Prozedur dann als Erweiterung an den bereits bestehenden Filterausdruck an, der dann etwa so aussieht:

```
BLZ = '35040123' AND Teilnehmer = '638415101234' AND Konto = '411891234'
```

Diesen Ausdruck weist die Prozedur schließlich der Eigenschaft *Filter* des Unterformulars im Unterformular-Steuerelement zu.

```
Private Sub cboAccounts_AfterUpdate()
    UnterformularAktualisieren
End Sub
```

### 5.2.3 Aktualisieren der Konten nach Kontakt-Auswahl

Die Auswahl eines Eintrags mit dem Kombinationsfeld *cboContacts* soll den Inhalt des Kombinationsfeldes *cboAccounts* entsprechend aktualisieren. Dies erledigt die Prozedur, die durch das Ereignis *Nach Aktualisierung* von *cboContacts* ausgelöst wird. Den Code dieser Prozedur sehen Sie hier:

```
Private Sub cboContacts AfterUpdate()
    Dim strAccounts As String
    If Not IsNull(Me!cboContacts) And Not Nz(Me!cboContacts, -1) = -1 Then
        strAccounts = GetAccountList(Me!cboContacts)
        strAccounts = "-1;;;;<Alle anzeigen>;" & strAccounts
        Me!cboAccounts.RowSource = strAccounts
        Me!cboAccounts = Me!cboAccounts.ItemData(0)
        Me!cboAccounts.SetFocus
        Me!cboAccounts.Dropdown
    Flse
        If Nz(Me!cboContacts) = 0 Then
            Me!cboAccounts.RowSource = ""
        Flse
            Me!cboAccounts.RowSource = "-1;;;;;<Alle anzeigen>;"
            Me!cboAccounts = Me!cboAccounts.ItemData(0)
        Fnd If
    Fnd If
   UnterformularAktualisieren
Fnd Sub
```

Die Prozedur arbeitet weitgehend wie die Prozedur *Form\_Open*. Der eine Unterschied ist, dass nach der Auswahl eines Eintrags mit *cboContacts* und der Einstellung der Datensatzherkunft des Kombinationsfeldes *cboAccounts* letzteres auch noch aufgeklappt wird (siehe Abbildung 5.8).

# 6 Umsätze einlesen

Dieses Kapitel zeigt, wie Sie die Umsätze eines Kontos in eine Tabelle Ihrer Datenbank einlesen.

Die nachfolgend beschriebene Vorgehensweise ist grundsätzlich nicht komplizierter als der Abruf des aktuellen Kontostands, aber ein wenig aufwendiger. Das hat zwei Gründe: Erstens gibt es diesmal nicht nur einen Satz von Rückgabewerten, sondern mehrere – und zwar je Umsatzposition einen. Und zweitens sollen die Umsätze nicht nur einfach im Formular angezeigt, sondern am besten gleich gespeichert werden – zum Beispiel in einer Tabelle.

Beispiele: Die Prozeduren dieses Kapitels finden Sie in den Modulen *mdlKontoauszug* und *cls-Transations*.

# 6.1 Ablauf beim Einlesen der Umsätze

Der grundsätzliche Ablauf sieht so aus:

- » Aufruf der Funktion Kontoauszug über eine Schaltfläche des Formulars frmUmsaetze
- » Abfrage der nötigen Zugangsdaten (etwa Passphrase bei Verwendung einer Schlüsseldatei oder PIN beim PIN/TAN-Verfahren)
- » Aufbau einer Verbindung zum Server der Bank
- » Zusammenstellen des Anfrage-Segments, hier HKKAZ
- » Ausführen des Segments
- » Auswerten der Antwort, die im Gegensatz zur Abfrage des Kontostands einige Informationen mehr zurückliefert
- » Speichern jeder einzelnen Umsatzposition in einem eigenen Objekt einer Auflistung
- » Auswerten der Auflistung, etwa durch Eintragen der gewonnenen Informationen in eine Tabelle mit den Umsätzen

## 6.2 Einlesen der Umsätze im Detail

Die nachfolgend beschriebenen Techniken sind für Einsteiger möglicherweise nicht allzu leicht nachzuvollziehen. Versuchen Sie es einfach – wenn Sie dies alles verstanden haben, können Sie vielleicht bald selbst mit der Klassenprogrammierung starten.

#### Kapitel 6 Umsätze einlesen

Beginnen wir mit einem kleinen Detail: Es gibt zwei Arten von Umsätzen, die man mit der Funktion *Kontoauszug* abrufen kann. Die erste Sorte sind die bereits gebuchten Umsätze, bei der zweiten handelt es sich um vorgemerkte Umsätze. Eines vorneweg: Wenn Sie beide gleichzeitig abfragen und in eine Tabelle schreiben, müssen Sie die Tabelle, in der die Umsätze gespeichert werden, unbedingt mit einem Feld versehen, das angibt, ob der gespeicherte Umsatz ein gebuchter oder ein vorgemerkter Umsatz ist.

Der Grund: Wenn Sie heute Ihre Umsätze einlesen, sind vielleicht drei Positionen dabei, die vorgemerkt sind. Diese landen aber auch in Ihrer Tabelle. Fragen Sie ein paar Tage später nochmals die Umsätze ab, sind die zuvor als vorgemerkt gekennzeichneten Umsätze gebucht und werden nochmals gespeichert.

Die Aufgabe ist nun, die Umsätze, die als vorgemerkt markiert waren, aus der Tabelle zu löschen, sobald sie gebucht wurden und somit zweimal in der Tabelle vorkommen. Dies ist aber nicht ganz trivial, weil Sie genau identifizieren müssen, für welchen vorgemerkten Umsatz bereits ein gebuchter Umsatz vorliegt. Daher werden wir in der Lösung dieses Kapitels ausschließlich mit gebuchten Umsätzen arbeiten und die noch nicht gebuchten, aber vorgemerkten Umsätze lediglich in einem Meldungsfenster ausgeben. Um der Funktion *Kontoauszug* mitzuteilen, ob Sie die gebuchten oder die vorgemerkten Umsätze abfragen wollen, haben wir eine Enumeration definiert. Eine Enumeration ist prinzipiell eine Variable, für die mehrere Werte zur Auswahl angeboten werden. Die Definition sieht so aus:

```
Public Enum eGebucht
    eUmsaetzeGebucht = 0
    eUmsaetzeNichtGebucht = 1
End Enum
```

Die Definition dieser Enumeration bringen Sie gemeinsam mit den anderen Deklarationen und Prozeduren im Standardmodul *mdlKontoauszug* unter. Schwenken wir nun direkt zur Funktion *Kontoauszug* über, wo diese Enumeration als Parameter verwendet wird. Der Kopf der Funktion mit den Parametern sieht so aus (die Klasse *clsTransactions* wird weiter unten beschrieben – diese müssen Sie noch implementieren, damit die Funktion *Kontoauszug* lauffähig wird):

```
Public Function Kontoauszug(intContact As Integer, _
    intAccount As Integer, _
    intGebucht As eGebucht, _
    objTransactions As clsTransactions, _
    Optional dateStart As Date, _
    Optional dateEnd As Date) _
    As Boolean
```

Die Parameter *intContact* und *intAccount* erwarten genau wie bereits die Funktion *Kontostand* den Index des Kontakts und des Kontos/Depots, für den die Umsätze bestimmt werden sollen. Diese liefert am einfachsten der Aufruf vom Formular *frmUmsaetze* aus, in dem Sie diese Werte

ja ganz einfach per Kombinationsfeld auswählen können. Der Parameter *intGebucht* hat den Typ der soeben vorgestellten Enumeration.

Abbildung 6.1 zeigt, wie sich diese Enumeration bei der Eingabe bemerkbar macht:

I	irektbereich	
	Kontoauszug 1, 1,	
	Kontoauszug( <i>intCon</i> : eUmsaetzeGebucht intGebucht / intGebucht As eGebucht, [dateStart As Date], [dateEnd As Date]) As else eUmsaetzenichtGebucht intGebucht intGebu	-



Als weitere Parameter können Sie das Startdatum und das Enddatum des Zeitraums eingeben, für den die Umsätze ermittelt werden sollen. Innerhalb der Funktion brauchen wir einige Variablen, die zu Beginn deklariert, aber erst in den folgenden Abschnitten erläutert werden:

Dim objAccountSegment As BACSegment Dim objDialog As BACDialog Dim objMessage As BACMessage Dim objSegmentAnfrage As BACSegment Dim objSegmentVersion As BACSegment Dim objSegmentVersion As BACSegment Dim objBACTransaction As BACTransaction Dim objLine As BACSwiftStatementLine Dim objContact As BACContact Dim objContact As BACContact Dim objAccount As BACAccount Dim objSwiftObject As Object Dim lngMaxSegmentVersion As Long Dim objTransaction As clsTransaction Dim strSegment As String

Die hier verwendeten Objektvariablen verwenden meist bereits per Verweis zum VBA-Projekt hinzugefügte Bibliotheken. Bei der Variablen *objLine* des Typs *BACSwiftStatementLine* ist dies nicht der Fall. Hier müssen Sie noch einen Verweis auf die Bibliothek *DataDesign DDBAC S.W.I.F.T. Format Component* hinzufügen (siehe Abbildung 6.2).

Kapitel 6 Umsätze einlesen



Abbildung 6.2: Verweis auf die Bibliothek DataDesign DDBAC S.W.I.F.T. Format Component

Die folgende Anweisung holt zunächst einen Verweis auf das benötigte Element der mit der Funktion *GetContacts* ermittelten Auflistung:

Set objContact = GetContacts(intContact)

Als Nächstes erscheint wiederum der Dialog, mit dem Sie die für die Anmeldung erforderlichen Daten eingeben – also etwa die Passphrase oder einen PIN:

```
Set objDialog = objContact.NewDialogUI
```

Sollte der Benutzer hier beispielsweise die *Abbrechen*-Schaltfläche betätigen, bleibt *objDialog* leer. Dies prüfen wir im nächsten Schritt, wobei die Funktion im Falle von *objDialog Is Nothing* beendet wird:

```
If objDialog Is Nothing Then
Kontoauszug = False
Exit Function
End If
```

Da wir diese Bezeichnung später mehrfach benötigen, speichern wir die Segmentkennung in einer Variablen:

```
strSegment = "HKKAZ"
```

Dann wird der Account festgelegt und ein Segment erstellt, aus dem die Prozedur später die Informationen zu dem Konto erhält, die für den Zugriff benötigt werden:

```
Set objAccount = objContact.Accounts(intAccount)
Set objAccountSegment = objAccount.Segment
```

Leseprobe! Vollständiges Buch erhältlich unter www.amvshop.de im André Minhorst Verlag.

Einlesen der Umsätze im Detail

Zuvor müssen wir allerdings noch Verbindung zum Bankserver aufnehmen, was die folgende Anweisung erledigt:

objDialog.BeginDialog

Das hier verwendete Segment heißt *HKKAZ*. Wie schon zuvor ermitteln wir die aktuellste Version des Segments und erzeugen ein neues Segment, das wir schließlich der Objektvariablen *objSegmentAnfrage* zuweisen.

In der folgenden Zeile ermitteln wir zunächst ein *BACSegment*-Objekt, das die optimale Version des Segments enthalten wird und speichern es in der Variablen *objSegmentVersion*:

```
Set objSegmentVersion = _
    objAccount.FindOptimalBPDSegment(strSegment, lngMaxSegmentVersion)
```

Dann ermitteln wir mithilfe der Eigenschaft *Version* des soeben ermittelten Segments *objSegmentVersion* die aktuellste Version des Segments *HKKAZ*:

Set objSegmentAnfrage = GetBanking.NewSegment(strSegment, objSegmentVersion.Version)

Das Objekt *GetBanking* liefert übrigens wie *GetContacts* oder *GetAccounts* eine kleine Funktion, die wie folgt aussieht und jeweils entweder einen bestehenden oder, falls nicht vorhanden, einen neuen Verweis auf ein Objekt des Typs *BACBanking* zurückgibt:

```
Public Function GetBanking() As BACBanking
    If m_Banking Is Nothing Then
        Set m_Banking = New BACBanking
    End If
    Set GetBanking = m_Banking
End Function
```

Nun füllt die Prozedur das *HKKAZ*-Objekt *objSegmentAnfrage* mit den benötigten Daten, die aus dem bereits erwähnten Account-Segment stammen:

```
objSegmentAnfrage("AuftraggeberKontoverbindung1", "Kontonummer1") = ______
objAccountSegment("Kontoverbindung1", "Kontonummer1")
objSegmentAnfrage("AuftraggeberKontoverbindung1", "Laenderkennzeichen1") = _____
objAccountSegment("Kontoverbindung1", "Laenderkennzeichen1")
objSegmentAnfrage("AuftraggeberKontoverbindung1", "Kreditinstitutcode1") = _____
objAccountSegment("Kontoverbindung1", "Kreditinstitutcode1")
objSegmentAnfrage("AuftraggeberKontoverbindung1", "Unterkontomerkma11") = ______
objAccountSegment("Kontoverbindung1", "Unterkontomerkma11") = ______
objAccountSegment("Kontoverbindung1", "Unterkontomerkma11")
objSegmentAnfrage("AlleKonten1") = False
```

Dies waren die Informationen, die wir bereits für die Abfrage des Kontostands eingegeben haben. Der vorliegende Aufruf benötigt aber noch weitere optionale Daten, nämlich den Zeitraum, für den die Abfrage die Umsätze liefern soll.

#### Kapitel 6 Umsätze einlesen

Dieser wird jedoch nur angegeben, wenn zumindest *dateStart* größer als *0* ist. Ist dann auch noch *datEnde* größer oder gleich *datStart*, erhalten die beiden Eigenschaften mit den Parametern *DatumVon1* und *DatumBis1* die entsprechenden Werte. Ist *datEnd* nicht größer oder gleich *datStart*, wird für den zweiten Parameter *DatumBis1* ebenfalls der Wert aus *datStart* angegeben:

```
If datStart > 0 Then
    If datEnde > 0 And (datEnde >= datStart) Then
        objSegmentAnfrage("DatumVon1") = datStart
        objSegmentAnfrage("DatumBis1") = datEnde
    Else
        objSegmentAnfrage("DatumVon1") = datStart
    End If
Else
        objSegmentAnfrage("DatumVon1") = DateAdd("d", -100, Date)
        objSegmentAnfrage("DatumBis1") = Date
End If
```

Die nächste Anweisung prüft zunächst den Status des Objekts objDialog:

If objDialog.State = bacDialogReady Then

Wenn dieser dem Wert der Konstanten *bacDialogReady* entspricht, kann die Ausführung des Segments starten. Dies erledigt wiederum die Methode *ExecuteSegment* des Objekts *objDialog*:

Set objMessage = objDialog.ExecuteSegment(objSegmentAnfrage)

Das Ergebnis landet im Objekt *objMessage*, und da es sich hier um eine einzelne Transaktion handelt, erhalten wir alle notwendigen Daten über das Element *Transactions(0)* von *objMessage*. Anschließend erfolgt eine Prüfung, ob die Transaktion genau ein Antwort-Segment geliefert hat:

```
Set objBACTransaction = objMessage.Transactions(0)
If objBACTransaction.ResponseSegments.Count = 1 Then
```

Ist dies der Fall, weisen wir das Antwort-Segment dem Objekt objSegmentAntwort zu:

Set objSegmentAntwort = objBACTransaction.ResponseSegments(0)

Hier kommt schließlich der Parameter *intGebucht* ins Spiel. In Abhängigkeit davon lesen wir nämlich das Antwort-Segment mit dem Parameterwert *UmsaetzeGebucht1* oder *Umsaetze-NichtGebucht1* in das Objekt *objSwiftObjekt* ein:

```
If Not IsEmpty(objSegmentAntwort) Then
   Select Case intGebucht
   Case 0
        Set objSwiftObject = objSegmentAntwort("UmsaetzeGebucht1")
   Case 1
```

#### Einlesen der Umsätze im Detail

```
Set objSwiftObject = objSegmentAntwort("UmsaetzeNichtGebucht1")
End Select
End If
Else
Exit Function
End If
```

*objSwiftObjekt* enthält die eigentlichen interessanten Daten der Antwort des Bankservers – die Umsätze. Betrachten Sie die nächsten Zeilen einfach so, als ob statt der *objTransction*...-Anweisungen links vom Gleichheitszeichen die Anweisung *Debug.Print* stehen würde und die gelieferten Informationen einfach im Direktfenster ausgegeben werden sollen – das erleichtert das Verständnis für den Moment, bevor wir uns im Anschluss an die hier tatsächlich angewandte Vorgehensweise heranwagen.

Das *objSwiftObject*-Objekt enthält eine Auflistung namens *StatementLines*, welche die folgenden Zeilen durchlaufen. Jedes Element der *StatementLines*-Auflistung wird dabei in das Objekt *objLine* geschrieben, das den Typ *BICSwiftStatementLine* hat. Dieses wiederum hat eine ganze Reihe von Eigenschaften, die genau den Informationen zu einer einzelnen Umsatzposition entspricht. So liefert *Amount* den Betrag der Transaktion, *EntryDate* das Buchungsdatum und *Purpose* den Verwendungszweck. Schauen Sie sich die in den folgenden Anweisungen enthaltenen Eigenschaften im Hinblick auf die Daten an, die Sie möglicherweise einmal für eigene Zwecke benötigen, und machen Sie sich dann auf einen Ausflug in die Welt der Klassenprogrammierung bereit:

```
Set obiTransactions = New clsTransactions
For Each objLine In objSwiftObject.StatementLines
   With objLine
        Set objTransaction = New clsTransaction
        objTransaction.BLZ = objContact.UserID
        objTransaction.Buchungsdatum = .ValidityDate
        objTransaction.Betrag = .Amount
        objTransaction.Verwendungszweck = .Purpose
        objTransaction.Auftraggeber = .Name
        objTransaction.Kundennummer = objContact.UserID
        objTransaction.Kontonummer =
            objContact.Accounts.Item(intAccount).AccountNumber
        objTransaction.BLZ Partner = .BankCode
        objTransaction.Kontonummer Partner = .AccountNumber
        objTransaction.Waehrung = .CurrencyCode
        objTransaction.BusinessTransactionCode =
            .BusinessTransactionCode
        objTransaction.BusinessTransactionText =
            .BusinessTransactionText
```

#### Kapitel 6 Umsätze einlesen

```
objTransaction.ChargesAmount = .ChargesAmount
       objTransaction.ChargesCurrencyCode = .ChargesCurrencyCode
       objTransaction.CustomerReference = .CustomerReference
       objTransaction.EquivalentAmount = .EquivalentAmount
       objTransaction.EquivalentCurrencyCode =
            .EquivalentCurrencyCode
       objTransaction.InstitutionReference = .InstitutionReference
       objTransaction.OriginalAmount = .OriginalAmount
       objTransaction.OriginalCurrencyCode = .OriginalCurrencyCode
       objTransaction.PrimaNoteNumber = .PrimaNoteNumber
       objTransaction.RealValidityDate = .RealValidityDate
       objTransaction.Reversal = .Reversal
       objTransaction.SupplementaryDetails = .SupplementaryDetails
       objTransaction.TextKeySupplement = .TextKeySupplement
       objTransaction.TransactionType = .TransactionType
       objTransaction.ValidityDate = .ValidityDate
       objTransactions.Add objTransaction
   Fnd With
Next objLine
```

Vorher noch ein kurzer Blick auf das Ende der Funktion: Diese gibt den Wert *True* als Funktionswert zurück. Die aufrufende Routine kann dann mit dem als Parameter übergebenen Objekt *objTransactions* die komplette Sammlung der Umsätze (mehr dazu weiter unten) auswerten. Anschließend wird noch der mit *objDialog* referenzierte Dialog beendet. Und für den Fall, dass keine Verbindung hergestellt werden konnte, findet sich auch noch eine *MsgBox*-Anweisung, die den Benutzer über das Scheitern des geplanten Vorgangs informiert.

```
Kontoauszug = True
objDialog.EndDialog
Else
MsgBox "Es konnte keine Verbindung hergestellt werden. "
& "Bitte synchronisieren Sie den Account erneut."
End If
End Function
```

### 6.3 Die Klasse clsTransactions

Natürlich könnten wir die Funktion *Kontoumsätze* auch einfach dazu verwenden, die Attributwerte der einzelnen Kontoumsätze in die im Anschluss vorgestellte Tabelle zu schreiben. Das ist aber unflexibel: Vielleicht hat ja jemand mal etwas ganz anderes mit diesen Daten vor und möchte sie beispielsweise in eine Textdatei oder in eine XML-Datei schreiben.
#### Die Klasse clsTransactions

Daher verwenden wir zwei Klassen, mit denen wir die Umsatzdaten temporär speichern und in Form eines einzigen Objekts an die aufrufende Prozedur zurückgeben, die dann mit den Daten machen kann, was sie will. In unserem Beispiel schreibt diese die Daten natürlich in eine Tabelle.

Schauen wir uns zunächst an, wie die beiden Klassen aufgebaut sind, die den möglicherweise umfangreichen Satz von Umsatzdaten in einem einzigen Objekt speichern. Um das Beispiel zu reproduzieren, legen Sie zunächst ein neues, leeres Klassenmodul an (VBA-Editor, Menüeintrag *Einfügen|Klassenmodul*) und speichern es unter dem Namen *clsTransactions*. Dieses Klassenmodul ist der Bauplan für Objekte, die sämtliche Daten einer Umsatzposition speichern können.

Wir schauen uns im Folgenden für Lernzwecke zunächst eine Minimalversion der Klasse an, damit Klassen-Einsteiger leichter nachvollziehen können, wie diese programmiert und eingesetzt wird.

Legen Sie in der frisch erzeugten Klasse eine private Variable namens *m\_Verwendungszweck* an. Diese soll einmal den Verwendungszweck für die in diesem Objekt gespeicherte Umsatzposition enthalten:

Dim m Verwendungszweck As String

Zusätzlich benötigen wir zwei *Property*-Prozeduren, eine mit dem Schlüsselwort *Set* und eine mit *Get*. Die erste dient dazu, der privaten Variablen *m\_Purpose* einen Verwendungszweck zuzuweisen und hat deshalb auch einen Parameter namens *strVerwendungszweck*, den die aufrufende Zeile füllen soll:

```
Public Property Let Verwendungszweck(strVerwendungszweck As String)
    m_Verwendungszweck = strVerwendungszweck
End Property
```

Mit der zweiten lässt sich dieser Verwendungszweck wieder abfragen:

```
Public Property Get Verwendungszweck() As String
    Verwendungszweck = m_Verwendungszweck
End Property
```

In einem Standardmodul deklarieren, instanzieren und verwenden Sie nun diese Rumpfversion der Klasse *clsTransactions* – die folgenden Zeilen dienen allerdings nur Beispielzwecken:

```
Public Sub Klassenbeispiel()
  Dim objTransaction As clsTransaction
  Set objTransaction = New clsTransaction
  With objTransaction
   .Verwendungszweck = "Testverwendungszweck"
  End With
  MsgBox "Verwendungszweck aus der Klasse: "
```

Kapitel 6 Umsätze einlesen

& objTransaction.Verwendungszweck

End Sub

Hier wird die Klasse zunächst als *objTransaction* deklariert und dann instanziert, das heißt, eine neue Instanz dieser Klasse wird erstellt.

Anschließend weist die Prozedur der Eigenschaft *Verwendungszweck* den Wert *Testverwendungszweck* zu und gibt diesen anschließend in einem Meldungsfenster aus – direkt als Eigenschaft des auf der Klasse *clsTransaction* basierenden Objekts, versteht sich. Auf die gleiche Art fügen wir noch eine Reihe weiterer Eigenschaften zur Klasse *clsTransaction* hinzu.

Die neben dem Verwendungszweck wichtigsten Eigenschaften wie die Kontonummer und die Bankleitzahl des Kontos, der Betrag und das Buchungsdatum sehen als Eigenschaften der Klasse *clsTransaction* so aus:

```
Dim m Verwendungszweck As String
Dim m Kontonummer As String
Dim m BLZ As String
Dim m Betrag As Currency
Dim m Buchungsdatum As Date
Dim m Auftraggeber As String
Dim m Kontonummer Partner As String
Dim m BLZ Partner As String
Dim m Waehrung As String
Public Property Get Verwendungszweck() As String
   Verwendungszweck = m Verwendungszweck
End Property
Public Property Let Verwendungszweck(strVerwendungszweck As String)
   m Verwendungszweck = strVerwendungszweck
End Property
Public Property Get Kontonummer() As String
   Kontonummer = m Kontonummer
End Property
Public Property Let Kontonummer(strKontonummer As String)
   m Kontonummer = strKontonummer
End Property
Public Property Get BLZ() As String
```

**Die Klasse clsTransactions** 

```
BLZ = m BLZ
End Property
Public Property Let BLZ(strBLZ As String)
   m BLZ = strBLZ
End Property
Public Property Get Betrag() As Currency
   Betrag = m Betrag
End Property
Public Property Let Betrag(curBetrag As Currency)
   m Betrag = curBetrag
End Property
Public Property Get Buchungsdatum() As Date
   Buchungsdatum = m Buchungsdatum
End Property
Public Property Let Buchungsdatum(datBuchungsdatum As Date)
   m Buchungsdatum = datBuchungsdatum
End Property
Public Property Get Auftraggeber() As String
   Auftraggeber = m Auftraggeber
End Property
Public Property Let Auftraggeber(strAuftraggeber As String)
   m Auftraggeber = strAuftraggeber
End Property
Public Property Get Kontonummer Partner() As String
   Kontonummer Partner = m Kontonummer Partner
End Property
Public Property Let Kontonummer Partner(strKontonummer Partner As String)
   m Kontonummer Partner = strKontonummer Partner
End Property
Public Property Get BLZ Partner() As String
   BLZ Partner = m BLZ Partner
```

#### Kapitel 6 Umsätze einlesen

```
End Property
Public Property Let BLZ_Partner(strBLZ_Partner As String)
    m_BLZ_Partner = strBLZ_Partner
End Property
Public Property Get Waehrung() As String
    Waehrung = m_Waehrung
End Property
Public Property Let Waehrung(strWaehrung As String)
    m_Waehrung = strWaehrung
End Property
```

Von hier aus können Sie noch einmal einen Blick zurück zur Beschreibung der Prozedur Kontoauszug werfen, und zwar dorthin, wo wir empfohlen haben, sich statt der Zuweisungen der eingelesenen Daten zu den Eigenschaften einer Klasse einfache *Debug.Print*-Anweisungen vorzustellen. Hier setzen wir nun, wo wir die Klasse *clsTransaction* kennen, auf der unser Objekt *objTransaction* basiert, erneut an. Hier nochmal, in gekürzter Fassung, der relevante Code:

```
Set objTransactions = New clsTransactions
For Each objLine In objSwiftObject.StatementLines
With objLine
Set objTransaction = New clsTransaction
objTransaction.BLZ = objContact.UserID
objTransaction.Buchungsdatum = .EntryDate
' ... Zuweisung vieler weiterer Eigenschaften
objTransactions.Add objTransaction
End With
Next objLine
```

Da wird zunächst ein Objekt *objTransactions* instanziert, das gleich alle erzeugten Objekte des Typs *clsTransaction* aufnehmen soll. Diese werden innerhalb der Schleife, die alle Umsatzpositionen im Element *objSwiftObject* durchläuft, erzeugt, mit Daten gefüllt und schließlich mit der *Add*-Methode zur Klasse *clsTransactions* hinzugefügt.

Nehmen wir das Objekt *objTransactions*, das die ganzen Objekte des Typs *clsTransaction* aufnimmt, zunächst als Black Box hin und schauen uns an, was sich damit anstellen lässt.

Tabelle zum Speichern der Umsatzdaten

# 6.4 Tabelle zum Speichern der Umsatzdaten

Die Umsatzdaten wollen wir in einer eigenen Tabelle speichern. Diese Tabelle legen Sie zunächst an und speichern sie unter dem Namen *tblUmsaetze*. Die Tabelle speichert nur die wesentlichen Felder – wenn Sie weitere benötigen, erweitern Sie einfach die Tabelle und die nachfolgend vorgestellten Zeilen zum Übertragen der Daten aus den Objekten des Typs *clsTransaction*. Die Tabelle sieht wie in Abbildung 6.3 aus, die Feldnamen und die zugeordneten Datentypen sind weitgehend selbsterklärend.

-8	frmOnlinbanking				23
۲	Bankverbindung:			•	
	Konto/Depot:	Commerzbank 35040038 638415109800 Postbank privat 76010085 937289858			
		DataDesign Demobank FinTS3 70000997 111	130	AL.	
Da	tensatz: 14 🖂 1 von 1	L > N > Kein Filter Suchen			

Abbildung 6.3: Entwurfsansicht der Tabelle tblUmsaetze

Ein Hinweis zur Zuordnung des Datentyps *Text* zu den Feldern, die Bankleitzahlen und Kontonummern speichern: Dies ist gewollt, weil diese gelegentlich führende Nullen aufweisen und diese bei Zahlen-Datentypen nicht angezeigt werden. Außerdem sollen die Werte bei Sortierungen nicht nach dem Zahlenwert, sondern in alphanumerischer Reihenfolge dargestellt werden.

### 6.4.1 Problem Abholzeitpunkt

Wir nehmen noch eine Überlegung vorneweg, die sich auf folgendes Problem bezieht: Sie können zwar genau festlegen, für welchen Zeitraum die Umsätze ermittelt werden sollen. Dies kann jedoch nur tagesgenau geschehen. Wenn Sie nun am 1. Februar 2014 mittags um 12.00 Uhr die Umsätze abfragen, erhalten Sie alle Umsätze bis zu diesem Zeitpunkt, unter Umständen also auch solche, die genau am 1. Februar 2014 gebucht wurden.

Wenn Sie nun am 2. Februar erneut die Umsätze einlesen möchten, und zwar genau diejenigen, die noch nicht eingelesen wurden, können Sie dies über die Angabe eines Zeitraums nicht zuverlässig erledigen: Wenn Sie alle Umsätze einlesen, die am 2. Februar 2014 erfolgten, fehlen möglicherweise welche, die am 1. Februar nach 12.00 Uhr mittags erfolgten. Wenn Sie als Beginn des Zeitraums hingegen den 1. Februar einstellen, erhalten Sie solche Umsätze, die bis 12.00 Uhr mittags gebucht wurden, ein zweites Mal.

### Kapitel 6 Umsätze einlesen

Das ist ein Problem, weil Sie ja keine Umsatzposition zwei Mal in der Tabelle *tblUmsaetze* speichern wollen. Es gibt nun zwei Möglichkeiten:

- » Hypothetisch: Sie rufen die Umsätze immer genau um 0:00 Uhr ab. Das ist aber nur praktikabel, wenn Sie diesen Vorgang fernsteuern oder um diese Zeit regelmäßig vor dem Rechner sitzen (ich weiß, dass das einige von Ihnen tun ...).
- » Sie prüfen jeweils beim Einlesen der Umsätze, ob die aktuelle Position bereits in der Tabelle gespeichert ist, und schreiben nur diese in die Tabelle, die noch fehlen. Hier gibt es allerdings das Problem, dass ja auch einmal zwei genau identische Umsätze bezogen auf die in der Tabelle gespeicherten Daten getätigt werden.
- » Die dritte Vorgehensweise ist die einfachste: Wenn Sie zuletzt am 1. Februar 2014 die Umsätze abgefragt haben (sagen wir um 12:00 Uhr) und nun am 2. Februar 2014 erneut um 12:00 Uhr die Umsätze abfragen, löschen Sie einfach zuvor alle Umsätze für den Tag, an dem Sie zuletzt die Umsätze abgefragt haben. Die Umsätze für diesen Tag liegen ja nun vollständig auf dem Bankserver vor, sodass Sie die in der Tabelle *tblUmsaetze* gespeicherten Umsätze für den 1. Februar 2014 leicht löschen und durch den kompletten Satz der Umsätze für diesen Tag ersetzen können.

Kehren wir nun zum Formular *frmUmsaetze* zurück und fügen diesem eine Schaltfläche namens *cmdUmsaetzeEinlesen* hinzu (siehe Abbildung 6.4).

🗐 frmUmsaetze			23
• · · · 1 · · 1 · · 2 · · · 3 · · · 4 · · · 5 · · · 6 · · · 7 · · · 8 · · · 9 ·	1 • 10 •	1 11	1 · 🔺
Formularkopf			
Umsätze einlesen			
✓ Detailbereich			
Bankverbindung: Ungebunden		-	
Konto/Depot: Ungebunden		•	
-			
✓ Formularfuß			_
1			

Abbildung 6.4: Neue Schaltfläche zum Abfragen der Umsätze

Die Ereigniseigenschaft *Beim Klicken* dieses Steuerelements statten wir mit der folgenden Ereignisprozedur aus (die in der *Execute*-Anweisung verwendete Funktion *SQLDatum* zur SQL-konformen Formatierung des Datums finden Sie im Modul *mdlTools* der Beispieldatenbank):

```
Private Sub cmdUmsaetzeEinlesen Click()
```

#### Tabelle zum Speichern der Umsatzdaten

```
Dim objTransactions As clsTransactions
Dim obiTransaction As clsTransaction
Dim db As DAO.Database
Dim i As Integer
Dim intNeueUmsaetze As Integer
Dim intGeloeschteUmsaetze As Integer
Dim datStart As Date
Dim datEnde As Date
Dim strKontonummer As String
Dim strBLZ As String
Set db = CurrentDb
strBLZ = Me!cboContacts.Column(2)
strKontonummer = Me!cboAccounts.Column(1)
datStart = Nz(DMax("Buchungsdatum", "tblUmsaetze", "Kontonummer = '"
   & strKontonummer & "' AND BLZ = '" & strBLZ & "'"), 0)
If DateDiff("d", datStart, Date) > 90 Then
   datStart = DateAdd("d", -90, Date)
Fnd If
If Kontoauszug(Me!cboContacts, Me!cboAccounts,
   eUmsaetzeGebucht, objTransactions, datStart, datEnde) = True Then
   If objTransactions.count > 0 Then
       datStart = objTransactions.Item(1).Buchungsdatum
       db.Execute "DELETE FROM tb]Umsaetze WHERE Buchungsdatum = "
            & SQLDatum(datStart), dbFailOnError
       intGeloeschteUmsaetze = db RecordsAffected
   End If
   For i = 1 To objTransactions.count
       With objTransactions.Item(i)
            db.Execute "INSERT INTO tblUmsaetze(Kundennummer, Kontonummer, BLZ, "
               & "Betrag, Waehrung, Buchungsdatum, Auftraggeber, "
                & "Kontonummer Partner, BLZ Partner, Verwendungszweck)"
               & " VALUES('" & .Kundennummer & "', '" & .Kontonummer & "', '"
                & .BLZ & "', " & Replace(.Betrag, ",", ".") & ", '" & .Waehrung _
                & "', " & SQLDatum(.Buchungsdatum) & ", '" & .Auftraggeber
                & "', '" & .Kontonummer Partner & "', '" & .BLZ_Partner _
                & "', '" & .Verwendungszweck & "')", dbFailOnError
            intNeueUmsaetze = intNeueUmsaetze + 1
       End With
   Next i
End If
If intNeueUmsaetze - intGeloeschteUmsaetze = 0 Then
```

### Kapitel 6 Umsätze einlesen

```
MsgBox "Es wurden keine neuen Umsätze eingelesen."

Else

MsgBox "Es wurden " & intNeueUmsaetze - ingGeloeschteUmsaetze _

& " neue Umsätze eingelesen."

End If

Me!sfmUmsaetze.Form.Requery

End Sub
```

Die Prozedur füllt nach dem Deklarationsteil zunächst die Objektvariable *db* mit einem Verweis auf das *Database*-Objekt der aktuellen Datenbank. Dieses brauchen wir später, um die zum Löschen der Umsätze des Datums mit den neuesten Umsätzen in der Tabelle *tblUmsaetze* und zum Hinzufügen der Umsatzposition nötigen Aktionsabfragen aufzurufen.

Dann ermitteln wir aus dem Kombinationsfeld *cboContacts* die Bankleitzahl der Bank zu dem Konto, dessen Umsätze eingelesen werden sollen, und speichern diese in der Variablen *strBLZ*. Auf ähnliche Weise beziehen wir die Kontonummer aus dem Kombinationsfeld *cboAccounts*. Diese landet in der Variablen *strKontonummer*.

Die anschließende *DMax*-Funktion ermittelt das Datum des neuesten Eintrags der Tabelle *tbl-Umsaetze* für die in *strBLZ* und *strKontonummer* gespeicherte BLZ und Kontonummer. Wie erwähnt: Die Daten für den zuletzt eingelesenen Tag wollen wir löschen, da für diesen Tag möglicherweise nicht alle Umsätze eingelesen wurden, und erneut einlesen. Dies ist allerdings auch nur sinnvoll, wenn dieser Tag überhaupt neu eingelesen werden kann. Und dadurch, dass man aktuell nur maximal bis zu 90 Tage zurückliegende Buchungen einlesen kann, müssen wir hier noch eine weitere Bedingung einbauen. Dabei prüft die Prozedur, ob das nun in *datStart* gespeicherte Datum mehr als 90 Tage zurückliegt. Ist dies der Fall, wird *datStart* neu gefüllt, und zwar mit dem Datum, das 90 Tage vor dem aktuellen Datum liegt.

Nun kommt die Funktion *Kontoauszug* zum Zuge, die wir ja bereits weiter oben beschrieben haben und die ein Objekt auf Basis der Klasse *clsTransactions* mit allen Umsätzen zurückliefert. Wenn überhaupt aktuelle Umsätze ermittelt werden konnten, ist das Ergebnis der Funktion Kontoauszug der Wert *True*. In diesem Falle werden die folgenden, in einer entsprechenden *If... Then*-Bedingung eingefassten Anweisungen ausgeführt.

Danach geht es rund: Die Prozedur prüft, ob *objTransactions* tatsächlich mindestens einen Eintrag enthält. Ist dies der Fall, speichert die Prozedur das Buchungsdatum des ersten enthaltenen Elements in der Variablen *datStart*. Die folgende *Execute*-Methode führt eine *DELETE*-Anweisung aus, die alle Datensätze aus der Tabelle *tblUmsaetze* löscht, die dieses Datum im Feld *Buchungsdatum* aufweisen. Die Anzahl der gelöschten Datensätze speichert die Prozedur in der Variablen *intGeloeschteDatensaetze*, deren Zweck weiter unten erläutert wird.

Dann ermittelt die Prozedur mit der Eigenschaft *Count* die Anzahl der in *objTransactions* enthaltenen Umsatzpositionen und durchläuft diese in einer *For...Next-*Schleife. Innerhalb dieser Schleife greift die Prozedur über die Referenz *Item(i)* mit *i* als Laufvariable auf alle in *obj-*

### Tabelle zum Speichern der Umsatzdaten

*Transactions* enthaltenen Objekte des Typs *clsTransaction* zu und schreibt den Inhalt einiger Eigenschaften in einen neuen Datensatz der Tabelle *tblUmsaetze*. Dies geschieht mit der *Execute*-Methode des *Database*-Objekts *db*.

Die *INSERT INTO*-Aktionsabfrage, die den Datensatz anfügt, wirkt etwas lang, ist aber schnell zu überblicken: Sie schreibt einfach die Werte der benötigten Eigenschaften von *objTransaction* in die entsprechenden Felder der Tabelle *tblUmsaetze*.

Die Prozedur *cmdUmsaetzeEinlesen\_Click* prüft nun noch die mit der Zählervariablen *intNeue-Umsaetze* ermittelte Anzahl der neuen Umsätze und subtrahiert hiervon die in *intGeloeschteDatensaetze* gespeicherte Anzahl der gelöschten Umsätze. Ist die Anzahl der hinzugefügten Umsätze größer als die der gelöschten, gibt die Prozedur mit einer Meldung die Anzahl der neu hinzugefügten Umsätze aus. Anderenfalls erscheint ein Hinweis, dass keine Datensätze neu hinzugefügt wurden. Zur Erinnerung: Wir löschen alle Umsätze für den ersten Tag, für den wir neue Umsätze einlesen, um zu verhindern, dass Umsätze doppelt eingelesen werden.

In jedem Fall aktualisiert die Prozedur die Datenherkunft des noch hinzuzufügenden Unterformulars *sfmUmsaetze* mit der *Requery*-Methode.

Bevor wir uns zum Einarbeiten der Kontoumsätze an das Formular *frmUmsaetze* machen, kümmern wir uns noch kurz um die Klasse *clsTransactions*. Diese hat immerhin die verantwortungsvolle Aufgabe, die Umsätze und ihre Eigenschaften wohlbehalten von der Funktion *Kontoauszug* zur aufrufenden Ereignisprozedur *cmdUmsaetzeAbfragen* zu transportieren. Außerdem erhalten Sie so wieder ein wenig Know-how rund um die Klassenprogrammierung.

### 6.4.2 Klasse zum Sammeln der Umsatzpositionen

Die Klasse *objTransactions* deklariert und instanziert gleich beim Öffnen ein *Collection*-Objekt namens *mColTransactions*. Objekte des Typs *Collection* sind dafür gedacht, ein oder mehrere Objektverweise aufzunehmen.

In unserem Fall wollen wir für jede Umsatzposition eine eigene Klasse anlegen, die mit ihren Eigenschaften (die wir natürlich alle selbst definieren können) die entsprechenden Eigenschaften der Umsatzposition aufnehmen kann. *mColTransactions* wird wie folgt im Kopf des neuen Klassenmoduls deklariert und instanziert:

Dim mColTransactions As New Collection

Die folgende Methode der Klasse heißt Add und erwartet einen Objektverweis auf ein Objekt des Typs clsTransaction als Parameter. Diesen schreibt sie gleich als neues Element in die Collection *mColTransactions* – mit der dort ebenfalls Add benannten Methode:

```
Public Sub Add(objTransaction As clsTransaction)
    mColTransactions.Add objTransaction
End Sub
```

Kapitel 6 Umsätze einlesen

Kassensturz gefällig? Die folgende Funktion liefert die Anzahl der in der Collection *mColTransactions* enthaltenen Elemente. Diesen Wert erhält sie über die Eigenschaft *Count* des Collection-Objekts:

```
Public Function Count()
   Count = mColTransactions.Count
End Function
```

Nach dem Einlesen fehlt noch eine Funktion, mit der eine Prozedur von außen auf die in der Collection gesammelten Objektverweise zugreifen kann, um auf die Attribute der entsprechenden *clsTransaction*-Objekte zuzugreifen.

Dies erledigt die Klasse *clsTransactions* mit der Funktion *Item*, die den Index des gewünschten Elements als Parameter erwartet und dieses aus der Collection *mColTransactions* herausliest und als Rückgabewert der Funktion zurückliefert:

```
Public Function Item(intIndex As Integer) As clsTransaction
   Set Item = mColTransactions.Item(intIndex)
End Function
```

## 6.5 Umsätze im Formular anzeigen

Nun müssen wir die in der Tabelle *tblUmsaetze* gespeicherten Umsätze noch im Formular *frm-Umsaetze* unterbringen – es wäre doch schön, wenn der Benutzer die frisch eingelesenen Umsätze gleich im Formular betrachten kann.

Dazu verwenden wir ein Unterformular namens *sfmUmsaetze*, das die Tabelle *tblUmsaetze* als Datenherkunft verwendet und deren Felder in der Datenblattansicht anzeigt. Dieses Formular fügen Sie nun als Unterformular zum Formular *frmUmsaetze* hinzu. Das Unterformular soll seine Daten in der Datenblattansicht anzeigen.

Das notwendige Unterformular *sfmUmsaetze* verwendet die Tabelle *tblUmsaetze* als Datenherkunft. Ziehen Sie in der Entwurfsansicht des Formulars alle benötigten Felder für die Übersicht in den Detailbereich des Formulars. Alle Felder müssen hier nicht erscheinen, sondern nur diejenigen, die für einen Überblick notwendig sind. Später werden wir eine Funktion zur Anzeige der Umsatzdetails in einem eigenen Formular hinzufügen. Damit der Benutzer nicht selbst Umsätze hinzufügen kann, stellen Sie die Eigenschaft *Anfügen zulassen* auf den Wert *Nein* ein. Das Unterformular sieht nun wie in Abbildung 6.5 aus.

#### Umsätze im Formular anzeigen



Abbildung 6.5: Unterformular zur Anzeige der Umsätze in der Entwurfsansicht

Stellen Sie noch die Eigenschaft *Standardansicht* auf *Datenblatt* ein, damit das Unterformular seine Daten in der Datenblattansicht anzeigt. Nun ziehen Sie das Unterformular in das Hauptformular, sodass dieses wie in Abbildung 6.6 aussieht. Stellen Sie außerdem die beiden Eigenschaften *Horizontaler Anker* und *Vertikaler Anker* jeweils auf den Wert *Beide* ein, damit sich das Unterformular ebenfalls vergrößert, wenn Sie das Hauptformular vergrößern.

Kapitel 6 Umsätze einlesen



Abbildung 6.6: Das Formular frmUmsaetze mit dem Unterformular sfmUmsaetze in der Entwurfsansicht

### 6.5.1 Multikontenfähigkeit

Im aktuellen Zustand zeigt das Formular immer alle Umsätze an (siehe Abbildung 6.7). Wir müssen den Inhalt des Unterformulars also noch nach den ausgewählten Werten der Kombinationsfelder *cboContacts* und *cboAccounts* filtern.

# 7 SEPA-Überweisung

Die Überweisung dürfte die meistverwendete Aktion sein, bei der tatsächlich Geld von Konto A nach Konto B wandert. Dementsprechend wollen wir uns diesem Thema etwas ausführlicher widmen.

# 7.1 Überweisungsformular verwenden

Die Beispieldatenbank stellt ein Formular namens *frmSEPAUeberweisung* bereit, dass alle für die Überweisung notwendigen Daten erfasst (siehe Abbildung 7.1).

😑 Überweisung		_		23		
SEPA-	Überweisung					
Auftraggeber:						
Bankverbindung:	10100		-			
Konto/Depot:	) Girokonto			-		
IBAN:	DDBADEMM002					
BIC:						
-Empfänger:						
Empfänger:	Andre Minhorst		-	1		
Bezeichnung:						
Empfängername 1:	Andre Minhorst					
Empfängername 2:						
IBAN:	HE CONSISTENCE IN CONTRACTOR					
BIC:						
-Sonstige Daten:						
Betrag:	0,01					
Verwendungszweck (max. 27 Zeichen/ Zeile):	Beispielverwendungszweck					
Überweisung am:	20.02.2014					
Meldung:						
Uberweisen						

Abbildung 7.1: Das Formular frmUeberweisung

#### Kapitel 7 SEPA-Überweisung

Die ersten Kombinationsfelder ermöglichen wie die bereits vorher vorgestellten Formulare die Auswahl des Kontaktes und des Kontos, von dem aus die Überweisung erfolgen soll. Nach der Auswahl des Kontos werden die beiden Textfelder mit der IBAN und der BIC des Kontos gefüllt. Aus diesem Grund zeigt das untere Kombinationsfeld auch nur solche Konten an, die SEPA-fähig sind. Als Zweites definiert der Benutzer den Empfänger der Überweisung. Dies geschieht mithilfe der Steuerelemente im zweiten Block des Formulars. Wenn zuvor noch keine Überweisungen getätigt wurden, gibt es noch keine gespeicherten Empfänger. Diese müssen dann für jeden neuen Empfänger in die vier Steuerelemente mit den Beschriftungen *Empfängername 1, Empfängername 2, IBAN* und *BIC* eingegeben werden. Aus *Empfängername 1* und *Empfängername 2* wird gleich bei der Eingabe eine *Bezeichnung* zusammengestellt, die im Textfeld darüber erscheint. Diese Bezeichnung kann nicht manuell angepasst werden und sollte eindeutig sein.

Der untere Block enthält schließlich die übrigen für die Überweisung wichtigen Daten:

- » Betrag: Zu überweisender Betrag, gegebenenfalls mit Komma als Dezimaltrennzeichen
- » Verwendungszweck: Verwendungszweck mit 27 Zeichen pro Zeile
- » Überweisung am: Datum der Überweisung
- » Meldung: Antwort der Bank auf die übergebene Überweisung, wird automatisch gefüllt

Nach dem Eingeben aller notwendigen Daten und dem Absenden mit der Schaltfläche Überweisen wird der Kontakt zur Bank hergestellt. Wenn das Absenderkonto beispielsweise mit PIN/ TAN arbeitet, erscheint nun der Dialog aus Abbildung 7.2.

] HBCI/FinTS-Kontakt	<b>×</b>
PIN (Passwort) Eingabe	
Für die gewünschte Funktion wird eine gültige P	IN (Passwort) benötigt.
Homebanking Kontakt: DataDesign Demobank FinTS3 #2	
TAN Verfahren:	Verfügbare TAN Medien:
902 mobile TAN 👻	TAN Liste: 1139282051 🔻
PIN (Passwort):	
•••••	
	4 5 6
	789
Zurück     Weiter	Abbrechen

Abbildung 7.2: Eingabe des PINs für die Anmeldung am Bankserver

#### Überweisungsformular verwenden

Nach der Eingabe wird beispielsweise der TAN an ein mobiles Endgerät gesendet. Dieser muss dann über einen weiteren Dialog eingegeben werden (siehe Abbildung 7.3).

rei-Schritt-TAN-Verfahren						
TAN - Eingabe						
Kontakt:	DataDesign Demobank FinTS3 #2	2 3				
Verfahren:	mobile TAN	5 6				
TAN:	123456 7	8 9				
Bitte beacht SMS Die TAN wu	Bitte beachten Sie folgende Hinweise: SMS Die TAN wurde an das Handy mit der Handynr. 0179-1000902 verschickt.					
		-				

Abbildung 7.3: Abfrage der TAN-Nummer, die an ein mobiles Endgerät geschickt wurde

Wurden PIN und TAN korrekt eingegeben, führt die Anwendung die Überweisung durch und die Antwort der Bank wird im Textfeld ganz unten angezeigt (siehe Abbildung 7.4).

🔳 Überweisung			23
EI SEPA	-Überweisung		
-Auftraggeber:			
Bankverbindung:	DataDesign Demobank FinTS3 #2 70000997	1	-
Überweisung am:	20.02.2014		
Meldung:	Der Auftrag wurde entgegengenommen.		

Abbildung 7.4: Die Überweisung wurde erfolgreich ausgeführt.

Kapitel 7 SEPA-Überweisung

# 7.2 Überweisungsformular erstellen

Das Formular *frmSEPAUeberweisung* sieht im Entwurf wie in Abbildung 7.5 aus. Es ist ungebunden, die oberen vier Steuerelemente werden gegebenenfalls mit Standarddaten gefüllt.

🗐 frmUeberweisung		- •	23
I · I · I · I · 2 · I · 3 ·	1 • 4 • 1 • 5 • 1 • 6 • 1 • 7 • 1 •	8 · I · 9 · I · 10 · I · 11 · I	• 12 · 📥
Formularkopf			
SEPA-	Überweisung		
Detailbereich			_
Auftraggeber:			
Bankverbindung:	Ungebunden	•	
- Konto/Depot:	Ungebunden	▼	
2. IBAN:	Ungebunden		
- BIC:	Ungebunden		
3			=
Empfänger:			
Empfänger:	Ungebunden	🗹	
Bezeichnung:	Ungebunden		
Empfängername 1:	Ungebunden		
6 Empfängername 2:	Ungebunden		
IBAN:	Ungebunden		
7 BIC:	Ungebunden		
- Sonstige Daten:			
Betrag:	Ungebunden		
Verwendungszweck	Ungebunden		
(max. 27 Zeichen/			
- Zeile):			
11			
Uberweisung am:	Ungebunden		
12 Meldung:	Ungebunden		
-			
13			
14 🛃 Überweisen [	3 ок		
Formularfuß			
4			
<u></u>			

Abbildung 7.5: Das Formular frmUeberweisung in der Entwurfsansicht

### 7.2.1 Kombinationsfelder füllen

Dazu löst das Formular beim Öffnen die Ereignisprozedur Form\_Open aus, die wie folgt aussieht:

```
Private Sub Form_Open(Cancel As Integer)
Dim intLastContact As Integer
```

Überweisungsformular erstellen

```
Dim intLastAccount As Integer
   Me!cboContacts.RowSourceType = "Value List"
   Me!cboAccounts.RowSourceType = "Value List"
   Me!cboContacts.RowSource = GetContactList
   If Len(Me!cboContacts.RowSource) = 0 Then
       MsgBox "Es wurden keine Kontakte angelegt."
       Cancel = True
       Exit Sub
   End If
   intLastContact = Nz(DLookup("LetzterBankkontakt", "tbl0ptionen"), -1)
   If Not intLastContact = -1 Then
       Me!cboContacts = Me!cboContacts.ItemData(intLastContact)
       Me!cboAccounts.RowSource = GetAccountList(intLastContact. True)
       intLastAccount = Nz(DLookup("LetztesKonto", "tbl0ptionen"), -1)
       If Not intLastAccount = -1 Then
           Me!cboAccounts = Me!cboAccounts.ItemData(intLastAccount)
           Me!txtBICSender = Me!cboAccounts.Column(3)
           Me!txtIBANSender = Me!cboAccounts.Column(2)
       End If
   End If
   Me!cboEmpfaenger = Me!cboEmpfaenger.ItemData(0)
End Sub
```

Die Prozedur stellt die Eigenschaft *Herkunftsart* beider Kombinationsfelder auf *Wertliste* ein (*RowSourceType/"Value List"*). Dann ermittelt sie mit der Funktion *GetContactList* die Liste aller verfügbaren Kontakte, die der Benutzer zuvor über die Systemsteuerung angelegt hat. Liegen noch keine Kontakte vor, bricht die Prozedur das Öffnen des Formulars an dieser Stelle mit einer entsprechenden Meldung ab.

Sollte der Benutzer in diesem oder in einem anderen Formular bereits zuvor einmal einen Kontakt ausgewählt haben, ist dieser in der Tabelle *tblOptionen im* Feld *LetzterBankkontakt* gespeichert. Diesen Wert liest die Prozedur aus und stellt den entsprechenden Wert im Kombinationsfeld *cboContacts* ein. In diesem Fall ermittelt die Prozedur mit der Funktion *GetAccount-List* auch gleich noch die Liste der Kontos dieses Kontakts und weist diese Liste der Eigenschaft *RowSource* des Kombinationsfeldes *cboAccounts* zu. *GetAccountList* wird in diesem Fall mit dem Wert *True* für den zweiten, optionalen Parameter *bolSepa* aufgerufen. Dies führt dazu, dass nur SEPA-fähige Konten zurückgeliefert werden.

Auch beim Kombinationsfeld *cboAccounts* könnte gegebenenfalls bereits der zuletzt verwendete Wert in der Tabelle *tblOptionen* gespeichert sein. In diesem Fall wählt die Prozedur den entsprechenden Eintrag des Kombinationsfeldes aus und führt zwei weitere Anweisungen aus. Die erste füllt das Textfeld *txtBlCSender* mit dem Inhalt der dritten Spalte des Kombinationsfeldes Kapitel 7 SEPA-Überweisung

und trägt somit die *BIC* ein, die zweite füllt das Textfeld *txtIBANSender* mit dem Inhalt der zweiten Spalte des Kombinationsfeldes und somit mit dem *IBAN* des überweisenden Kontos.

### 7.2.2 Das Kombinationsfeld zur Auswahl der Empfänger

Schließlich stellt die Prozedur noch das Kombinationsfeld *cboEmpfaenger* auf den ersten verfügbaren Wert ein. Dieses enthält aber doch zu Beginn noch gar keine Datensätze? Kein Problem: Damit der Benutzer dennoch weiß, was er mit diesem Steuerelement anfangen soll, erstellen wir eine eigene Tabelle, welche dem Kombinationsfeld einen Eintrag mit einem entsprechenden Hinweis füllt, der übrigens immer als erster Eintrag beim Öffnen des Formulars erscheinen soll.

Das Kombinationsfeld soll ganz oben einen Eintrag mit dem Text <*Empfänger auswählen oder unten eingeben>* anzeigen. Dafür verwenden wir eine *UNION*-Abfrage, die mit dem oberen Teil diesen generischen Datensatz liefert und mit dem unteren Teil die tatsächlich vorhandenen Datensätze aus der Tabelle *tblEmpfaenger*:

```
SELECT 0 AS EmpfaengerID, '<Empfänger auswählen oder unten eingeben>'
AS Bezeichnung, '' AS IBAN, '' AS BIC
FROM tblEmpfaenger
UNION
SELECT EmpfaengerID, [Name1] & " " & [Name2] AS Bezeichnung, IBAN, BIC
FROM tblEmpfaenger
ORDER BY Bezeichnung;
```

Das Dumme ist nur: Wenn die Tabelle *tblEmpfaenger* keine Datensätze enthält, liefert auch diese *UNION*-Abfrage ein leeres Ergebnis. Also erstellen wir schnell eine Dummy-Tabelle, die genau so aufgebaut ist wie die Tabelle *tblEmpfaenger*, aber *tblEmpfaengerBasis* heißt und genau einen Datensatz mit beliebigen Daten enthält (siehe Abbildung 7.6).

	🔠 tblEmpfaengerBasis 🗆 🗉 🖾									
2	EmpfaengerID 👻	Bezeichnung 👻	Name1 🝷	Name2 👻	IBAN 👻	BIC 👻				
	1	bla	1	1	1	1				
*	(Neu)									
Da	Datensatz: H									

Abbildung 7.6: Dummy-Tabelle für die Anzeige eines Datensatzes im Kombinationsfeld

Dementsprechend müssen wir noch die UNION-Abfrage so anpassen, dass sich der erste Teil auf die Tabelle *tblEmpfaengerBasis* bezieht:

```
SELECT 0 AS EmpfaengerID, '<Empfänger auswählen oder unten eingeben>'
AS Bezeichnung, '' AS IBAN, '' AS BIC
FROM tblEmpfaengerBasis
```

Überweisungsformular erstellen

UNION ...

Nun zeigt das Kombinationsfeld *cboEmpfaenger* immer mindestens den Eintrag mit dem Text <*Empfänger auswählen oder unten eingeben>* an – unabhängig davon, ob die Tabelle *tblEmpfaenger* Daten enthält oder nicht.

### 7.2.3 Empfänger bei Überweisung speichern

Wenn die Überweisung ausgeführt wird, soll der aktuell eingegebene Empfänger in der Tabelle *tblEmpfaenger* gespeichert werden. Dies erledigt eine Funktion, die beim Anklicken der Schaltfläche *cmdUeberweisen*, auf die wir später zurückkommen, aufgerufen wird. Diese Funktion sieht so aus:

```
Private Function EmpfaengerSpeichern() As Boolean
   Dim db As DAO.Database
   Dim lngError As Long
   Set db = CurrentDb
   If IsNull(DLookup("Bezeichnung", "tblEmpfaenger", "Bezeichnung = '"
           & Me!txtBezeichnung & "' AND Name1 = '" & Me!txtName1 & "' AND Name2 = '"
           & Me!txtName2 & "' AND IBAN = '" & Me!txtIBANEmpfaenger & "' AND BIC = '"
           & Me!txtBICEmpfaenger & "'")) Then
       On Error Resume Next
       db.Execute "INSERT INTO tblEmpfaenger(Bezeichnung, Name1, Name2, IBAN, BIC) "
           & "VALUES('" & Me!txtBezeichnung & "', '" & Me!txtName1 & "', '"
           & Me!txtName2 & "', '" & Me!txtIBANEmpfaenger & "', '"
           & Me!txtBICEmpfaenger & "')", dbFailOnError
        lngError = Err.Number
       On Error GoTo 0
       Select Case IngError
           Case 3022
                If MsgBox("Es ist bereits ein Empfänger gleichen Namens vorhanden. "
                       & "Überschreiben (Ja) oder ändern (Nein)?", vbYesNo +
                       vbExclamation, "Empfänger vorhanden") = vbYes Then
                   db.Execute "UPDATE tblEmpfaenger SET Name1 = '" & Me!txtName1
                       & "', Name2 = '" & Me!txtName2 & "', IBAN = '"
                       & Me!txtIBANEmpfaenger & "', BIC = '" & Me!txtBICEmpfaenger
                       & "' WHERE Bezeichnung = '" & Me!txtBezeichnung & "'",
                       dbFailOnError
                   EmpfaengerSpeichern = True
               Else
                   With Me!txtBezeichnung
                       .Enabled = True
```

### Kapitel 7 SEPA-Überweisung

```
.Locked = False
.SetFocus
End With
Exit Function
End If
Case 0
EmpfaengerSpeichern = True
Case Else
MsgBox "Fehler " & lngError & ": '" & AccessError(lngError) & "'"
End Select
Else
EmpfaengerSpeichern = True
End If
End Function
```

Die Funktion prüft zunächst, ob bereits ein identischer Datensatz in der Tabelle *tblEmpfaenger* vorliegt. Ist dies der Fall, geschieht nichts weiter – die Funktion liefert lediglich den Wert *True* zurück. Anderenfalls deaktiviert die Prozedur für die Ausführung der folgenden Anweisung die Fehlerbehandlung und versucht dann, einen neuen Datensatz in der Tabelle *tblEmpfaenger* mit den Werten der Textfelder *txtBezeichnung*, *txtName1*, *txtName2*, *txtIBAN* und *txtBIC* anzulegen. Dies kann zu einem Fehler führen, wenn der Wert des Textfeldes *txtBezeichnung* bereits in der Tabelle enthalten ist. Die Fehlernummer wird dann in der Variablen *IngError* gespeichert und die Fehlerbehandlung wieder aktiviert. Ist der Inhalt des Feldes *txtBezeichnung* noch nicht in der Tabelle enthalten, speichert die Funktion den Datensatz einfach. In diesem Fall gibt die Funktion den Wert *True* zurück.

Ist der Wert von *txtBezeichnung* jedoch bereits vorhanden, hat *IngError* den Wert *3022*, was die Anzeige eines Meldungsfensters nach sich zieht, das den Benutzer fragt, ob der den vorhandenen Datensatz mit gleicher Bezeichnung überschreiben oder den Datensatz mit einer neuen Bezeichnung erstellen möchte. Im ersten Fall aktualisiert die Prozedur die Felder des betroffenen Datensatzes mit den neue Daten, im zweiten Fall wird die Funktion abgebrochen und der Fokus auf das Textfeld *txtBezeichnung* gesetzt, dass nun auch zur Bearbeitung freigegeben wird.

Die Funktion wird an dieser Stelle beendet und gibt den Wert *False* an die aufrufende Routine zurück.

### 7.2.4 Neuen Empfänger eingeben

Wenn Sie einen neuen Empfänger eingeben möchten, der noch nicht in der Tabelle *tblEmpfaen*ger gespeichert ist, verwenden Sie die dazu vorgesehen Felder *txtName1*, *txtName2*, *txtlBAN-Empfaenger* und *txtBICEmpfaenger* dazu. Das Feld *txtBezeichnung* soll lediglich die Inhalte der Felder *txtName1* und *txtName2* zusammenfassen und somit den im Kombinationsfeld *cboEmpfaenger* anzuzeigenden Wert bereitstellen. Warum zeigen wir dann nicht direkt die Inhalte der

### Überweisungsformular erstellen

Felder Name1 und Name2 im Kombinationsfeld an? Weil es ja den gleichen Empfänger durchaus mehrmals mit verschiedenen Konten geben kann. Für diesen Fall sehen wir vor, dass die Bezeichnung abweichend von Name1 und Name2 geändert werden kann.

Damit die Bezeichnung in *txtBezeichnung* zunächst einmal mit den Inhalten von *txtName1* und *txt-Name2* übereinstimmt, haben wir für das Ereignis *Bei Änderung* dieser beiden Textfelder jeweils eine Ereignisprozedur hinterlegt, welche eine weitere Routine namens *BezeichnungAnpassen* aufrufen und den eigenen Text sowie den Inhalt des jeweils anderen Textfeldes übergeben:

```
Private Sub txtName1_Change()
   BezeichnungAnpassen Me!txtName1.Text, Nz(Me!txtName2)
End Sub
Private Sub txtName2_Change()
   BezeichnungAnpassen Nz(Me!txtName1), Me!txtName2.Text
End Sub
```

Die Routine *BezeichnungAnpassen* nimmt diese Werte per Parameter entgegen und trägt den konkatenierten Wert in das Feld *txtBezeichnung* ein:

```
Private Sub BezeichnungAnpassen(strName1 As String, strName2 As String)
   Me!txtBezeichnung = Trim(strName1 & " " & strName2)
End Sub
```

### 7.2.5 Empfänger auswählen

Wenn der Benutzer bereits Überweisungen getätigt hat, kann er die bisher verwendeten Empfänger über das Kombinationsfeld *cboEmpfaenger* anzeigen lassen und auswählen (siehe Abbildung 7.7).



Abbildung 7.7: Auswahl eines bereits gespeicherten Empfängers

#### Kapitel 7 SEPA-Überweisung

Die Auswahl eines der Einträge des Kombinationsfeldes löst das Ereignis Nach Aktualisierung aus, für die Sie die folgende Ereignisprozedur hinterlegen. Diese fragt die Daten der Tabelle *tblEmpfaenger* ab und trägt die entsprechenden Feldwerte in die Textfelder *txtBezeichnung*, *txt-*Name1, *txtName2*, *txtIBANEmpfaenger* und *txtBICEmpfaenger* ein:

```
Private Sub cboEmpfaenger_AfterUpdate()
Me!txtBezeichnung = DLookup("Bezeichnung", "tblEmpfaenger", _
    "EmpfaengerID = " & Me!cboEmpfaenger)
Me!txtName1 = DLookup("Name1", "tblEmpfaenger", _
    "EmpfaengerID = " & Me!cboEmpfaenger)
Me!txtName2 = DLookup("Name2", "tblEmpfaenger", _
    "EmpfaengerID = " & Me!cboEmpfaenger)
Me!txtIBANEmpfaenger = DLookup("IBAN", "tblEmpfaenger", _
    "EmpfaengerID = " & Me!cboEmpfaenger)
Me!txtBANEmpfaenger = DLookup("BAN", "tblEmpfaenger", _
    "EmpfaengerID = " & Me!cboEmpfaenger)
Me!txtBICEmpfaenger = DLookup("BIC", "tblEmpfaenger", _
    "EmpfaengerID = " & Me!cboEmpfaenger)
Me!txtBICEmpfaenger = DLookup("BIC", "tblEmpfaenger", _
    "EmpfaengerID = " & Me!cboEmpfaenger)
```

Nachfolgend lernen Sie noch ein Formular kennen, dass alle in der Tabelle *tblEmpfaenger* gespeicherten Datensätze zur Bearbeitung anzeigt. Dieses öffnen Sie vom Formular *frmSEPAUeberweisung* aus mit einem Mausklick auf die Schaltfläche *cmdEmpfaengerBearbeiten* und die dadurch ausgelöste Ereignisprozedur:

```
Private Sub cmdEmpfaengerBearbeiten_Click()
    DoCmd.OpenForm "frmEmpfaenger", WindowMode:=acDialog
End Sub
```

# 7.3 Empfänger bearbeiten

Da die Anwendung alle eingegebenen Empfänger speichert, möchte der Benutzer gegebenenfalls hin und wieder die Einträge bearbeiten oder löschen. Vielleicht gibt es aber auch eine Liste von Empfängern etwa aus einem anderen Programm, dass der Benutzer der Anwendung in die Tabelle *tblEmpfaenger* importieren möchte (nach Wunsch können Sie diese Tabelle natürlich auch noch erweitern).

Das Formular zum Bearbeiten der Empfänger sieht wie in Abbildung 7.8 aus. Es besteht aus einem Hauptformular namens *frmEmpfaenger* und dem Unterformular *sfmEmpfaenger*. Das Unterformular zeigt die Daten der Tabelle *tblEmpfaenger* in der Datenblattansicht an, das Hauptformular stellt die Steuerelemente zum Bearbeiten der Datensätze zur Verfügung.

Empfänger bearbeiten

Empfänger							
	Bezeichnung 🚽	Name1 🚽	Name2 -	IBAN	BIC -		
	Andre Minhorst	Andre	Minhorst	DE1670000997100073516	DDBADEMM002		
	Klaus Müller	Klaus	Müller	DE12121212232323233434	ABCDEFGMM001		
*							
Da	tensatz: 🖬 🖣 🖥 vo	n 3 → H	🛤 🛛 🕅 Keir	Filter Suchen			
Хок							

Abbildung 7.8: Das Formular frmEmpfaenger mit Beispieldatensätzen

### 7.3.1 Unterformular sfmEmpfaenger erstellen

Das Unterformular *sfmEmpfaenger* verwendet die Tabelle *tblEmpfaenger* als Datenherkunft, genau genommen eine Abfrage, die auf dieser Tabelle basiert. Diese sortiert die Datensätze nach dem Feld *Bezeichnung*. Die Abfrage sieht wie in Abbildung 7.9 aus.

📑 sfmEmpfae	enger : Abfrage-Gen	erator				_ 0	Σ3
tblEmpfae	nger						
Fempfa Bezeir	sengerID						
Name	1						
IBAN BIC							
							•
Feld: Tabelle:	EmpfaengerID tblEmpfaenger	Bezeichnung tblEmpfaenger	Name1 tblEmpfaenger	Name2 tblEmpfaenger	IBAN tblEmpfaenger	BIC tblEmpfaenger	
Sortierung:		Aufsteigend 💂					
Anzeigen:	<b>V</b>		<b>V</b>	<b>V</b>	<b>V</b>	<b>V</b>	
Kriterien:							
oder:							

Abbildung 7.9: Datenherkunft des Unterformulars sfmEmpfaenger

Nachdem Sie die Datenherkunft festgelegt haben, können Sie alle Felder der Abfrage außer *EmpfaengerID* in den Detailbereich des Formularentwurfs ziehen. Das Ergebnis sieht dann etwa wie in Abbildung 7.10 aus.

Kapitel 7 SEPA-Überweisung

sfmEmpfaenger	• 3 • 1 • 4 • 1 • 5 • 1 • 6 • 1 • 7 • 1 •	 □ X3
	Bezeichnung Name1 Name2 IBAN BIC	
4 	III	•

Abbildung 7.10: Das Unterformular *sfmEmpfaenger* in der Entwurfsansicht

Dieses Formular ziehen Sie dann in das zuvor mit den wesentlichen Steuerelementen versehene Hauptformular *sfmEmpfaenger*. Abbildung 7.11 zeigt den Entwurf des Hauptformulars mit integriertem Unterformular.

-8	frmE	mpfaenger					23
	+ 1	• 1 • 1 • 2 • 1 • 3	4 5 6	• 7 • 1 • 8 • 1 • 9 • 1 • 10 • 1 • 11 •	1 + 12 + 1 + 13	· · · 14	•
	<b>₹</b> F	ormularkopf					
· - 1	2	Empf	änger				
	€ D	etailbereich					_
1	Emp	ofänger:					=
1			1 • 3 • 1 • 4 • 1 • 5 • 1 • 1	5 • 1 • 7 • 1 • 8 • 1 • 9 • 1 • 10 • 1	· 11 · 1 · 12 ·	1	
-	✓ Detailbereich						
2	:	Bezeichnung:	Bezeichnung				
-	1	Name1:	Name1				
3.	1	Name2:	Name2				
-	2	IBAN:	IBAN				
-	1	BIC:	BIC				
5	3						
-		1			[		
H	<b>€</b> F	ormularfuß					
	-	OK					-

Abbildung 7.11: Haupt- und Unterformular in der Entwurfsansicht

Das Unterformular verwendet die Klasse *clsColumnWidth*, um die Spaltenbreiten optimal an die Inhalte anzupassen.

Empfänger bearbeiten

### 7.3.2 Überweisungsdaten

Der untere Bereich des Formulars *frmSEPAUeberweisung* enthält weitere Steuerelemente zur Aufnahme wichtiger Informationen:

- » txtBetrag: Zu überweisender Betrag.
- » txtVerwendungszweck: Verwendungszweck, der nicht länger als 140 Zeichen sein darf.
- » txtUeberweisungsdatum: Wenn eine Terminüberweisung geplant ist, können Sie hier das Überweisungsdatum eintragen.

Hier fügen wir noch die eine oder andere Prüfung ein.

### Überweisungbetrag prüfen

Es kann zum Beispiel vorkommen, dass der Benutzer bei der Eingabe des Betrags statt eines reinen Zahlenwertes noch die Währung (etwa *EUR*) mit angibt. In diesem Fall soll eine entsprechende Meldung erscheinen. Oder er verwendet ein anderes Zeichen als das Komma als Dezimaltrennzeichen. Dann soll ein entsprechender Hinweis angezeigt werden und der Fokus auf dem Feld *txtBetrag* verbleiben.

Dazu fügen wir die folgende Ereignisprozedur zum Klassenmodul des Formulars hinzu, die durch das Ereignis *Vor Aktualisierung* des Textfeldes *txtBetrag* ausgelöst wird:

```
Private Sub txtBetrag_BeforeUpdate(Cancel As Integer)

If Not IsNumeric(Me!txtBetrag) Then

MsgBox "Bitte geben Sie einen numerischen Wert ein.", _
vb0KOnly + vbExclamation, "Ungültiger Wert"
Cancel = True
Exit Sub
End If
If InStr(1, Me!txtBetrag, ".") > 0 Then
MsgBox "Bitte verwenden Sie das Komma als Dezimaltrennzeichen. Der Betrag darf " _
"keine Punkte enthalten.", vb0KOnly + vbExclamation, "Ungültiger Wert"
Cancel = True
Exit Sub
End If
End If
End If
End Sub
```

Wenn der Benutzer nun einen Wert wie *100.00* eingibt, erscheint eine entsprechende Meldung (siehe Abbildung 7.12).

# 8 SEPA-Terminüberweisungen

Terminüberweisungen sind ein Sonderfall der Überweisung. Zwar wird auch hier Geld von Konto A nach Konto B bewegt, allerdings gibt der Auftraggeber hier einen festen Termin an, zu dem die Überweisung durchgeführt werden soll. Damit gehen weitere Aktionen einher, denn im Gegensatz zu einer normalen SEPA-Überweisung, die sofort ausgeführt wird, kann man eine in Auftrag gegebene SEPA-Terminüberweisung noch löschen, ändern oder auch nur einsehen. Dieses Kapitel zeigt den Umgang mit Terminüberweisungen.

# 8.1 Formular zum Ausführen von Terminüberweisungen

Das zum Auswählen des Senders und zum Eintragen oder Auswählen des Empfängers verwendete Formular entspricht dem im Kapitel *»SEPA-Überweisung« ab Seite 251* vorgestellten Formular. Dieses enthält ein Textfeld namens *txtUeberweisungsdatum*, mit dem der Benutzer das geplante Datum eintragen kann (siehe Abbildung 8.1). Die Techniken, um die Kombinationsfelder dieses Formulars zu füllen et cetera finden Sie im oben genannten Kapitel.

Nach der Eingabe der für die Überweisung notwendigen Daten klickt der Benutzer auf die Schaltfläche *cmdUeberweisung*. Dies löst die nachfolgend auszugsweise abgedruckte Prozedur aus, die bereits vollständig im Kapitel *»SEPA-Überweisung« ab Seite 251* abgebildet und erläutert wurde. Interessant ist hier der letzte Teil, in dem sich entscheidet, ob eine einfache SEPA-Überweisung oder eine Terminüberweisung durchgeführt wird:

```
Private Sub cmdUeberweisen_Click()
...
If IsNull(Me!txtUeberweisungsdatum) Then
    Me!txtMeldung = SEPAUeberweisung(objKontakt, objSender, objEmpfaenger, _
        curBetrag, strVerwendungszweck)
Else
    datUeberweisung = Nz(Me!txtUeberweisungsdatum)
    Me!txtMeldung = TerminierteSEPAUeberweisung(objKontakt, objSender, _
        objEmpfaenger, curBetrag, strVerwendungszweck, datUeberweisung)
End If
End Sub
```

In letzterem Fall muss das Textfeld *txtUeberweisungsdatum* gefüllt sein, was in der *lf...Then*-Bedingung geprüft wird. Innerhalb der Bedingung ruft die Prozedur *cmdUeberweisung\_Click* dann die Funktion *TerminierteSEPAUeberweisung* auf. Kapitel 8 SEPA-Terminüberweisungen

🗄 Überweisung				23			
SEPA	-Überweisung						
Auftraggeber:							
Bankverbindung:			•				
Konto/Depot:			-				
IBAN:	DE43700009971000735160						
BIC:	DDBADEMM002						
Empfänger:							
Empfänger:	Klaus Müller		-	1			
Bezeichnung:	Klaus Müller						
Empfängername 1:	Klaus						
Empfängername 2:	mpfängername 2: Müller						
IBAN:	AN: DE16700009971000735161						
BIC:	DDBADEMM002						
Sonstige Daten:							
Betrag:	1						
Verwendungszweck:	Terminueberweisung Test 1						
(noch 115 Zeichen)							
Überweicung am:	22.2.2014						
ouer wersung utt.	23.2.2014						
Merdung.	Der Auftrag wurde ausgerunnt.						
🛃 Überweisen	🔀 ОК						

Abbildung 8.1: Terminüberweisung mit benutzerdefiniertem Datum

# 8.2 Terminüberweisung ausführen

Diese Funktion sieht wie folgt aus und unterscheidet sich an mehreren Stellen von der Funktion *SEPAUeberweisung*, die wir bereits im Kapitel »SEPA-Überweisung« ab Seite 251 abgebildet haben. Sie finden diese Funktion im Modul *mdIOBMA\_SEPATerminueberweisungen*:

Public Function TerminierteSEPAUeberweisung(objContact As BACContact,  $\_$ 

objSender As BACAccount, objEmpfaenger As BACAccount, curBetrag As Currency, \_ strVerwendungszweck As String, datUeberweisung As Date, lngAuftragID As Long) \_ As String

- Dim objBanking As BACBanking
- Dim objSegmentAnfrage As BACSegment
- Dim objSegmentAntwort As BACSegment
- Dim objSegmentVersion As BACSegment

#### Terminüberweisung ausführen

```
Dim objSepaMessage As BACSepaMessage
Dim obiSepaOrder As BACSepaOrder
Dim strError As String
Dim i As Integer
Dim objMessage As BACMessage
Dim objBACDialog As BACDialog
Dim objTransaction As BACTransaction
Dim strAntwort As String
Dim strAuftragsidentifikation As String
Dim db As DAO.Database
Dim strSegment As String
strSegment = "HKCSE"
Set objSegmentVersion = objSender.FindOptima1BPDSegment(strSegment, 0)
If Not objSegmentVersion Is Nothing Then
    Set objBanking = New BACBanking
    Set objSegmentAnfrage = objBanking.NewSegment(strSegment,
        objSegmentVersion.Version)
    Set objSepaMessage = New BACSepaMessage
    objSepaMessage.Segment = objSegmentAnfrage
    objSepaMessage.FromAccount = objSender
    Set objSepaOrder = objSepaMessage.Orders.Add
    objSepaOrder.ToAccount = objEmpfaenger
    objSepaMessage.ExecutionDate = datUeberweisung
    With objSepaOrder
        Amount = curBetrag
        .CurrencyCode = "EUR"
        .Purpose = strVerwendungszweck
    End With
    objSepaMessage.Verify strError
    If Len(strError) > 0 Then
        MsgBox strError
    Fnd If
    Set objBACDialog = objContact.NewDialogUI
    objBACDialog.BeginDialog
    Select Case objBACDialog.HBCIResultCode
        Case Is >= 9000
            TerminierteSEPAUeberweisung = objBACDialog.HBCIResult & " ("
                & objBACDialog.HBCIResultCode & ")"
            Exit Function
        Case Else
    Fnd Select
```

#### Kapitel 8 SEPA-Terminüberweisungen

```
Set ob.iMessage = ob.iBACDialog.ExecuteSegment(ob.iSegmentAnfrage)
       obiMessage.Acknowledgement.SaveAs CurrentProject.Path & "\result.txt"
       Set objTransaction = objMessage.Transactions(0)
       objTransaction.Acknowledgement.SaveAs CurrentProject.Path
           & "\Acknowledgements.txt"
       objTransaction.OrderSegment.SaveAs CurrentProject.Path & "\OrderSegment.txt"
       objTransaction.ResponseSegments.SaveAs CurrentProject.Path
           & "\ResponseSegments.txt"
       Select Case objTransaction.Acknowledgement.Item(2, 1)
           Case Is >= 9000
               TerminierteSEPAUeberweisung = objTransaction.Acknowledgement.Item(2, 3)
                    & "(" & objTransaction.Acknowledgement.Item(2, 1) & ")"
               Exit Function
           Case Else
       Fnd Select
        If objTransaction.ResponseSegments.count > 0 Then
           Set objSegmentAntwort = objTransaction.ResponseSegments(0)
           strAuftragsidentifikation = objSegmentAntwort("Auftragsidentifikation1", 1)
           For i = 1 To 100
               If Len(objTransaction.Acknowledgement.Item(i, 3)) > 0 Then
                    strAntwort = strAntwort & objTransaction.Acknowledgement.Item(i, 3)
               Fnd If
           Next i
           Set db = CurrentDb
           db.Execute "INSERT INTO tblAuftraege(IBANAuftraggeber, "
               & "BICAuftraggeber, NameEinsEmpfaenger, NameZweiEmpfaenger, "
               & "IBANEmpfaenger, BICEmpfaenger, Betrag, Verwendungszweck, "
               & "Auftragsdatum, Auftragsidentifikation) "
               & "VALUES('" & objSender.IBAN & "', '" & objSender.BIC & "', '"
               & objEmpfaenger.NameEins & "', '" & objEmpfaenger.NameZwei & "', '"
               & objEmpfaenger.IBAN & "', '" & objEmpfaenger.BIC & "'. "
               & Replace(curBetrag, ",", ".") & ", '" & strVerwendungszweck & "', "
               & SQLDatum(datUeberweisung) & ", '" & strAuftragsidentifikation & "')",
               dbFailOnError
            lngAuftragID = db.OpenRecordset("SELECT @@IDENTITY", dbOpenDynaset).Fields(0)
           TerminierteSEPAUeberweisung = strAntwort
       End If
       ob.jBACDialog.EndDialog
   Fnd If
End Function
```

Die Funktion erwartet die folgenden Parameter:

### Terminüberweisung ausführen

- » objContact: Objektverweis auf den verwendeten Kontakt
- » objSender: Verweis auf das BACAccount-Objekt für das sendende Konto
- » objEmpfaenger: Verweis auf das BACAccount-Objekt für das empfangende Konto
- » curBetrag: Zu überweisender Betrag
- » strVerwendungszweck: Verwendungszweck der Überweisung
- » datUeberweisung: Geplantes Datum der Überweisung
- » IngAuftragID: Rückgabeparameter, der die gegebenenfalls von der Bank zurückgelieferte Referenznummer liefert

Die Funktion ermittelt zunächst die optimale Version des in *strSegment* angegebenen Segments des Typs *HKCSE* (Terminierte SEPA-Überweisung einreichen) für das aktuelle Konto. Sollte dies vorhanden sein, erstellt sie das entsprechende Segment dieses Typs und speichert es in der Variablen *objSegmentAnfrage*. Dann erstellt sie ein neues Objekt des Typs *BACSepaMessage*, wobei es sich um das Objekt zum Übermitteln der Nachricht an den Bankserver handelt. Dieses Objekt weist eine Eigenschaft namens *Segment* auf, der Sie das in *objSegmentAnfrage* gespeicherte Objekt zuweisen. Außerdem legen Sie für die Eigenschaft *FromAccount* mit *objSepaMessage* referenzierte Objekts den Absender der Überweisung fest, indem Sie den in *objSender* enthaltenen Verweis auf das entsprechende, per Parameter übergebene *BACAccount*-Objekt angeben.

Danach erstellt die Prozedur ein neues Objekt vom Typ *BACSepaOrder*, und zwar über die *Add*-Methode der *Orders*-Auflistung des *BACSepaMessage*-Objekts. Dieses nimmt über die Eigenschaft *ToAccount* den Empfänger der Zahlung entgegen. Der zu überweisende Betrag, die Währung sowie der Verwendungszweck landen in den Eigenschaften *Amount*, *CurrencyCode* und *Purpose* des *BACSepaOrder*-Objekts. Allein das Ausführungsdatum wird dem *BACSepaMessage*-Objekt zugewiesen, nämlich für die Eigenschaft *ExecutionDate*.

Die Verify-Methode des BACSepaMessage-Objekts objSepaMessage prüft die Daten und liefert für die Variable strError eine entsprechende Meldung zurück, sollte ein Fehler gefunden werden. Die anschließende *If...Then*-Bedingung prüft, ob strError einen Text enthält und gibt den Inhalt gegebenenfalls in einer Meldung aus.

Schließlich startet die eigentliche Übermittlung der Überweisungsdaten an den Bankserver. Mit der *NewDialogUI*-Methode des *BACContact*-Objekts ruft die Prozedur den Dialog für die Anmeldung am Bank-Server auf, beispielsweise durch Eingabe des PIN-Codes (siehe Abbildung 8.2). Kapitel 8 SEPA-Terminüberweisungen

PIN (Passwort) Eingabe									
Für die gewünschte Funktion wird eine gültige PIN (Passwort) benöti									
Homebanking Kontakt:									
DataDesign Demobank FinTS3 #2									
TAN Verfahren:	Verfügbare TAN Medien:								
902 mobile TAN 👻	TAN Liste: 1139282051								
PIN (Passwort):									
	1 2 3								
	4 5 6								
	7 8 9								
Weiter	Abbrechen								

Abbildung 8.2: Eingabe des PIN-Codes zur Anmeldung am Bank-Server

Das Auslösen der *BeginDialog*-Methode übermittelt den PIN an die Bank und wartet das Ergebnis ab. Dieses fragen Sie über die beiden Eigenschaften *HBCIResultCode* und *HBCIResult* ab. *HBCIResultCode* liefert einen Wert größer oder gleich 9000, wenn ein Fehler auftritt. Dies werten wir in einer *Select Case*-Anweisung aus und beenden die Funktion im Falle eines Fehlers, wobei die Fehlerbeschreibung aus der Eigenschaft *HBCIResult* plus Fehlernummer als Funktionswert zurückgeliefert werden soll.

Danach startet der Dialog zum Bank-Server, wobei das Aufrufen der *ExecuteSegment*-Methode des *BACDialog*-Objekts den zweiten Dialog zur Eingabe des TAN-Codes hervorbringt – beispiels-weise zur Eingabe der per Handy empfangenen TAN-Nummer (siehe Abbildung 8.3).

Leseprobe! Vollständiges Buch erhältlich unter www.amvshop.de im André Minhorst Verlag.

Terminüberweisung ausführen

Kontakt:	DataDesign Demobank FinTS3 #2		3
Verfahren:	mobile TAN	4 5	6
TAN:		7 8	9
Bitte heacht	ten Sie folgende Hinweise		_
Bitte beacht SMS Die TAN wu	ten Sie folgende Hinweise: rde an das Handy mit der Handynr. 0179-100	10902 verschickt.	^

Abbildung 8.3: Eingabe der mobilen TAN

Wenn der Benutzer die TAN eingegeben hat, speichert die Prozedur den Inhalt des *Acknowled-gement*-Objekts in einer Textdatei (dies geschieht, um den Inhalt zu prüfen – Sie können diese Anweisung auch weglassen). Diese enthält beispielsweise eine Zeile wie die Folgende:

HIRMG:2:2+9050::Teilweise fehlerhaft.'

Dann speichert die Prozedur einen Verweis auf das erste Element der *Transactions*-Auflistung in der Variablen *objTransaction* (in diesem Fall gibt es nur eine Transaktion, aber Sie können bei anderen Geschäftsvorfällen auch mehrere Überweisungen gleichzeitig ausführen – zum Beispiel bei Sammelüberweisungen (die dieses Buch nicht behandelt) oder bei Sammellastschriften (siehe Kapitel »SEPA-Lastschriften verwalten« ab Seite 385).

Dieses Objekt enthält wiederum ein *Acknowledgement*-Objekt, dessen Elemente beispielsweise einen Ergebniscode und eine Ergebnismeldung bereithalten. Auch hier können Fehler auftreten, die sich durch Werte größer oder gleich *9000* auszeichnen.

Dieses Acknowledgement-Objekt können Sie wie auch das OrderSegment-Objekt und das ResponseSegments-Objekt als Textdatei speichern – und zwar innerhalb der Prozedur mit den folgenden Anweisungen:

objTransaction.Acknowledgement.SaveAs CurrentProject.Path & "\Acknowledgements.txt" objTransaction.OrderSegment.SaveAs CurrentProject.Path & "\OrderSegment.txt" objTransaction.ResponseSegments.SaveAs CurrentProject.Path & "\ResponseSegments.txt"

Die Datei OrderSegment.txt sieht etwa so aus:

#### Kapitel 8 SEPA-Terminüberweisungen

HKCSE:3:1+DE57760100850937289858:PBNKDEFF+urn?:iso?:std?:iso?:20022?:tech?:xsd?:pain.00 1.002.03+@1177@<?xml version="1.0" encoding="UTF-8"?><Document xmlns="urn:iso:std:iso:20 022:tech:xsd:pain.001.002.03" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi :schemaLocation="urn:iso:std:iso:20022:tech:xsd:pain.001.002.03 pain.001.002.03.xsd"><C stmrCdtTrfInitn><GrpHdr><MsgId>9184D30B-B5AE-47EF-858E-46B20EA261</MsgId><CreDtTm>2014-02-25T22:20:06</CreDtTm><Nb0fTxs>1</Nb0fTxs><CtrlSum>1.00</CtrlSum><InitgPty><Nm>Andre Minhorst</Nm></InitgPty></GrpHdr><PmtInf><PmtInfId>BFEF80EF-4B99-4CB5-A7A9-EC868FBB52<// PmtInfId><PmtMtd>TRF</PmtMtd><PmtTpInf><SvcLv1><Cd>SEPA</Cd></cre>/SvcLv1></PmtTpInf><ReqdE xctnDt>2014-02-26</ReqdExctnDt><Dbtr><Nm>Andre Minhorst</Nm></Dbtr><DbtrAcct><Id>ZDbtrAcct><DbtrAgt><FinInstnId><BIC>PBNKDEFF</BIC></fi></or>

Der Teil in einfachen Anführungszeichen im hinteren Teil der Datei entspricht der sogenannten *Pain*-Datei, welche die Auftragsdaten im XML-Format enthält. Diese haben wir hier etwas übersichtlicher dargestellt, wobei wir die wichtigsten Elemente fett hervorgehoben und bezeichnet haben:

```
<CstmrCdtTrfInitn>
    <GrpHdr>
        <MsaId>9184D30B-B5AE-47EF-858E-46B20EA261</MsaId>
        <CreDtTm>2014-02-25T22:20:06</CreDtTm>
        <Nb0fTxs>1</Nb0fTxs>
        <CtrlSum>1.00</CtrlSum> ->Betrag
        <InitgPty><Nm>Andre Minhorst</Nm></InitgPty>
    </GrpHdr>
    <PmtInf>
        <PmtInfId>BFFF80FF-4B99-4CB5-A7A9-FC868FBB52</PmtInfId>
        <PmtMtd>TRF</PmtMtd>
        <PmtTpInf>
            <SvcLvl><Cd>SEPA</Cd></SvcLvl>
        </PmtTpInf>
        <RegdExctnDt>2014-02-26</RegdExctnDt> ->Termin
        <Dbtr><Nm>Andre Minhorst</Nm></Dbtr> ->Sender
        <DbtrAcct>
            <1d>
                <IBAN>DE57760100850937289858</IBAN> ->Sender IBAN
            </ ] d>
        </DbtrAcct>
```

Terminüberweisung ausführen

```
<DbtrAqt>
            <FinInstnId>
                <BIC>PBNKDEFF</BIC> ->Sender BIC
            </FinInstnId>
        </DbtrAat>
        <ChrqBr>SLEV</ChrqBr>
        <CdtTrfTxInf>
        <PmtId><EndToEndId>NOTPROVIDED</EndToEndId></PmtId>
            <Amt>
                <InstdAmt Ccy="EUR">1.00</InstdAmt> ->Währung und Betrag
            </Amt>
            <CdtrAqt>
                <FinInstnId>
                    <BIC>COBADEFFXXX</BIC> ->BIC Empfänger
                </FinInstnId>
            </CdtrAat>
            <Cdtr>
                <Nm>Andre Minhorst</Nm>
            </Cdtr>
            <CdtrAcct>
                <1d>
                    <IBAN>DE24350400380415109800</IBAN> ->IBAN Empfänger
                </Id>
            </CdtrAcct>
            <RmtInf>
                <Ustrd>Test5</Ustrd> ->Verwendungszweck
            </RmtInf>
        </CdtTrfTxInf>
    </PmtInf>
</CstmrCdtTrfInitn>
```

Die Datei Acknowledgements.txt enthält beispielsweise eine Zeile wie die folgende – hier wurde soeben das *mTan*-Verfahren gesperrt:

HIRMS:3:2:3+9941::TAN ungültig, mTAN-Verfahren wurde vorläufig gesperrt.'

Die Prüfung, ob der Vorgang erfolgreich an die Bank übergeben werden konnte, erfolgt anhand eines Elements des *Acknowledgement*-Objekts (*objTransaction.Acknowledgement.Item(2, 1)*). Dieses liefert wiederum den Zahlencode für das Ergebnis des Vorgangs. Ist dieser größer oder gleich *9000*, ist – wie oben bereits angedeutet – ein Fehler aufgetreten. In diesem Fall wird die Funktion wiederum beendet und die Fehlermeldung samt Code als Funktionswert zurückgegeben.

Kapitel 8 SEPA-Terminüberweisungen

War der Vorgang hingegen erfolgreich, gibt das Kreditinstitut in der Regel einen Zahlenwert zurück, mit dem Sie den Vorgang – in diesem Fall eine Terminüberweisung – referenzieren können. Dies erlaubt beispielsweise das Prüfen des Vorhandenseins dieses Vorgangs oder auch das Löschen oder Ändern. Dazu erstellen wir ein Objekt des Typs *BACSegment* namens *objSegment*-*Antwort* (mit dem Typ *HICSE* – dies entspricht dem Geschäftsvorfall *Einreichung terminierter SEPA-Überweisung bestätigen*). Dieses Segment liefert dann für die Parameter "*Auftrags-identifikation*" und 1 den entsprechenden Zahlenwert, der in der String-Variablen *strAuftrags-identifikation* landet. Den Rest der Antwort entnimmt die Prozedur den Zeilen des *Acknowled-gement*-Objekts in einer Schleife über alle Zeilen.

Damit wir später auf die so erzeugte Überweisung zugreifen können, um diese beispielsweise zu löschen, speichern wir die Daten mit einer per *Execute*-Methode abgesetzten *INSERT INTO*-Anweisung in einer Tabelle namens *tblAuftraege*. Diese sieht im Entwurf wie in Abbildung 8.4 aus.

	Fe	Idname	Felddaten	typ	Beschreibung	1		
Þ	AuftragID		AutoWert		5			
	IBANAuftra	geber	Text					
	BICAuftrage	eber	Text					
	NameEinsEr	npfaenger	Text					
	Name7weiF	mnfaenger	Text					
	IBANEmofae	nger	Text					
	BICEmpfaon	aor	Text					
	Botrag	801	Währung					
	Vorwondun	rezwoek	Toxt					
	Auftragedate	gszweck	Detum/Ubracit					
	Auttragsdat	um -+:f:l+:	Datum/Unrzeit					
	Auttragside	ntifikation	lext			-,		
_						[		
			Feldeigensch	haften		_		
	Allgemein N	lachschlagen						
F	eldgröße	Long Inte	ger					
N	leue Werte	Inkrement	ŧ					
F	ormat							
Beschriftung Indiziert Ja (Ohne Du Smarttags Textausrichtung Standard			Devel (Header)					
			Duplikate)	Ein E	Feldname kann bis zu 64 Zeichen lang			
				sein, e	einschließlich Leerzeichen. Drücken Sie			
				F1,	um Hilfe zu Feldnamen zu erhalten.			

Abbildung 8.4: Entwurf der Tabelle tblAuftraege

Nach dem Hinzufügen einiger Aufträge sieht die Tabelle in der Datenblattansicht wie in Abbildung 8.5 aus.

#### Terminüberweisungen abrufen

	tblAuftraege												
2	AuftragID 👻	IB.	ANAuftraggeber 🔹	BICAuftraggeber 👻			meEinsEmp	ofaenger 👻	NameZweiEmp	faenger 👻 🔪			
	2 DE43700009971000735160 D				DDBADEMM002		Klaus		Müller	DET			
	3 DE43700009971000735160 0 4 DE43700009971000735160 0			DDBADEMM002 DDBADEMM002		Klaus Klaus		Müller	D				
								Müller	D.C.				
*	(Neu)												
ŀ		-									_		23
E		r. +	IBANEmpfaenger	Ŧ	BICEmpfaenge	r 👻	Betrag 🝷	Verwend	lungszweck 🕞	Auftragsdatum 👻	Auftragsidentif	ikatior	n -
E	DE1670000997100073				DDBADEMM00	2	1,00€	Terminue	perweisung Test	23.02.2014	142391		
Di	atensatz: I4 → 1	ensatz: H 4 1 vo DE1670000997100073		5161 DDBADEMM00		2	1,00€	Test4		26.02.2014	)14 143278		
		3	DE1670000997100073	85161	DDBADEMM00	2	1,00€	Test5		26.02.2014	143279		
		1											
		2											
		ξ											

Abbildung 8.5: Die mit ein paar Beispieldatensätzen gefüllte Tabelle tblAuftraege

Nach dem Speichern des Auftrags in der Tabelle ermittelt die Prozedur noch den Wert des Primärschüsselfeldes des neu angelegten Datensatzes und legt diesen in der als Rückgabevariablen definierten Variablen *IngAuftragID* ab. Die Methode *EndDialog* des *BACDialog*-Objekts beendet schließlich die Kommunikation mit der Bank.

# 8.3 Terminüberweisungen abrufen

Auf den ersten Blick scheint es eine gute Idee zu sein, gleich beim Übergeben einer Terminüberweisung einen entsprechenden Datensatz in einer Tabelle anzulegen. Durch die Speicherung der Auftragsidentifikation können Sie nachträglich auf die Terminüberweisungen zugreifen und diese einsehen, löschen oder ändern. Allerdings geht dies natürlich auch nur so lange, bis diese ausgeführt sind. Da kann man dann auch gleich den aktuellen Bestand der noch offenen Terminüberweisungen von der Bank abrufen und mit diesen arbeiten.

### 8.3.1 Terminüberweisungen verwalten

Nachdem wir Informationen über die Terminüberweisung in der Tabelle *tblAuftraege* gespeichert haben, wollen wir damit auch weitere Aktionen durchführen – zum Beispiel, um eine Terminüberweisung zu löschen oder zu ändern. Dazu legen wir zunächst ein Formular an, das in einem Unterformular die Terminüberweisungen anzeigt. Das Hauptformular heißt *frmTerminueberweisungen*, das Unterformular *sfmTerminueberweisungen*.

Im Hauptformular finden Sie die üblichen beiden Kombinationsfelder zur Auswahl des Kontakts und des Kontos. Das Unterformular soll die Datensätze der Tabelle *tblAuftraege* anzeigen und nach dem ausgewählten Konto gefiltert werden. Im Entwurf sieht das Formular wie in Abbildung 8.6 aus.
Kapitel 8 SEPA-Terminüberweisungen



Abbildung 8.6: Das Formular frmSEPATerminueberweisungenVerwalten in der Entwurfsansicht

Die beiden Kombinationsfelder im oberen Bereich arbeiten wie bei den bereits vorgestellten Formularen. Eine Änderung ist, dass sowohl das Laden des Formulars als auch das Aktualisieren des Kombinationsfeldes *cboAccounts* die Prozedur *IBANUndBICAktualisieren* aufruft. Diese trägt den IBAN und die BIC in die beiden Textfelder *txtIBAN* und *txtBIC* ein:

```
Private Sub IBANUndBICAktualisieren()
Dim objAccount As BACAccount
Set objAccount = GetAccounts(Me!cboContacts).Item(Me!cboAccounts)
With objAccount
        Me!txtIBAN = .IBAN
        Me!txtBIC = .BIC
    End With
End Sub
```

Damit das Unterformular immer die zum aktuell ausgewählten Konto gehörenden Terminüberweisungen anzeigt, sind die beiden Eigenschaften *Verknüpfen von* und *Verknüpfen nach* des Unterformular-Steuerelements auf die Werte *IBANAuftraggeber* und *txtIBAN* eingestellt.

# 9 SEPA-Daueraufträge

Daueraufträge richten Sie ein, wenn Sie periodisch wiederholbare Zahlungen auf das gleiche Konto überweisen möchten. Dazu gibt es unter SEPA einige Segmente, die Sie sich entweder als Vorbereitung für die Programmierung über die Dokumentation der DDBAC aus der Datei *SEPA Segmente.html* zusammensuchen können – oder Sie verwenden dazu das Formular *frmSegmente*, das wir weiter vorn vorgestellt haben.

Dort wählen Sie wie in Abbildung 9.1 zunächst das Konto aus, von dem aus Sie den Dauerauftrag testweise beim Programmieren ausführen möchten. Dann klicken Sie auf die Schaltfläche *Segmente prüfen* und aktivieren die Option *Nur verfügbare Elemente*. Dies führt zur Anzeige einiger weniger Segmente samt den für das aktuell markierte Segment verfügbaren Parameter.

== frmSegmente									23
🕉 💻 Verfü	gbare	Segment	e an	zeigen					
Bankverbindung:	Postban	k 76010085 9	37289	858	-	🔽 НК	V DK		
Konto/Depot:	937289	858   Postbank	Giro pl	us	•	<b>▼</b> HI	V DI		
Suche nach Segmen	tname od	er Kennung: da				V Nur	verfügbare Elen	nente	
≤= Segmente prüfe		d <sup>e</sup> Seg	gment-(	Code erstellen					
🕗 Verfügbar 🚽	Segme	entkennung	<b>-</b>		Segme	entname			Se
	HKCDB	U	SEPA	-Dauerauftrag	bestand at	orufen			
<b>V</b>	HKCDE		SEPA	Dauerauftrag	einrichten				
<b>V</b>	HKCDL		SEPA	A-Dauerauftrag	löschen				
	HKCDN		SEPA Dauerauftrag ändern						
*									
Datensatz: I4 → 1 Segmentelemente:	von 4 🕨	N 🕅 🏹 G	efiltert	Suchen	4	III			Þ
Position				Bezeichnung			Form	nat	
2		AuftraggeberK	ontove	rbindung			DEG		
2,1		AuftraggeberK	ontove	rbindung,IBAN			an(34)		≡
2,2	2.2 AuftraggeberKor		ontove	rbindung,BIC			an(11)		
2,3		AuftraggeberK	ontove	ntoverbindung,Kontonummer			id		
2,4		AuftraggeberKontove		overbindung,Unterkontomerkmal		id			
2,5		AuftraggeberKontoverbindung,Laenderkennzeichen			hen	ctr			
2,6		AuftraggeberKontoverbindung,Kreditinstitutcode			e	an(30)		_	
3 SEPAPainMessages				DEG					
Datensatz: 🛚 🚽 1	von 12	🕨 M M 🖬 🔣 K	ein Filt	er Suchen					
🔀 ок									

Abbildung 9.1: SEPA-Segmente für Dauerträge heraussuchen

Wenn nach dem Aktivieren der Option *Nur verfügbare Elemente* keine Segmente mehr angezeigt werden, kann das aktuelle Konto keine SEPA-Daueraufträge verwalten – um bei der Pro-

grammierung sinnvoll zu testen, sollten Sie also ein Konto wählen, dass dieses Feature unterstützt.

Achtung: Bei den verfügbaren Segmenten prüft das Formular nur diejenigen Segmente, die mit *HK...* beginnen. Die korrespondierenden *HI...*-Antwortsegmente setzen wir als verfügbar voraus.

Für die Beispiele dieses Kapitels betrachten wir die folgenden Segmente:

- » HKCDB: SEPA-Dauerauftragsbestand abrufen
- » HKCDE: SEPA-Dauerauftrag einrichten
- » HKCDL: SEPA-Dauerauftrag löschen

Nun können wir uns anschauen, wie das Segment für das Absenden eines Dauerauftrags aussieht. Leider lässt sich dies nicht so einfach programmieren wie in den vorherigen Beispielen: Wir müssen uns also ein wenig umgewöhnen – dazu später mehr.

# 9.1 Informationen für das Anlegen eines Dauerauftrags

Für einen Dauerauftrag müssen Sie verschiedene Informationen abfragen und übergeben. Die erste ist die Wahl des Intervalls. Es stehen die folgenden Intervalle zur Verfügung:

- » Wochen und
- » Monate.

Wenn Sie sich für eine Woche als Intervall entscheiden, legen Sie den Wochentag fest, an dem die Überweisung ausgeführt werden soll. Außerdem geben Sie an, ob der Dauerauftrag jede Woche erledigt werden soll oder alle wieviel Wochen dies geschehen soll.

Im Falle des monatlichen Intervalls geben Sie den Tag der Ausführung an, also beispielsweise 1, 2 oder 3. Auch hier können Sie festlegen, alle wieviel Monate der Dauerauftrag ausgeführt werden soll. Wichtig: Das Enddatum muss bezüglich des Ausführungstages auch wieder mit dem Tag des Startdatums korrespondieren. Wenn Sie also am 1.7.2014 starten und zwei Ausführungen wünschen, müssen Sie den 1.8.2014 als Enddatum angeben. Bei der Postbank sieht das Formular für die Eingabe der Daten für einen Dauerauftrag etwa wie in Abbildung 9.2 aus.

Wenn wir die Eingabemaske nachbauen möchten, benötigen wir einige Informationen vom jeweiligen Kreditinstitut. Dort erfahren wir beispielsweise, wie viele Tage Vorlaufzeit beim Einrichten des Dauerauftrags erlaubt sind oder welche Wochentage beim Anlegen von wöchentlichen Daueraufträgen zulässig sind. Gleiches gilt für die Monatstage bei der Verwendung von monatlichen Daueraufträgen. All diese Informationen liefert uns die Bank über das BACBankData-Objekt, das wir zunächst ermitteln müssen

Empfänger Aus Vorlagen wählen			
Name:	IBAN:		
	IBAN nicht bekannt		
Überweisungsdaten			
Betrag: 0,00 €	VerwZweck:		
Optionen Standardoptionen Alle Optionen			
Intervall			
Alle 1 Monate		Erste Ausführung am:	?
• immer am 3		Aufhebungsdatum:	?
🔵 Immer am Letzten des Mon	ats	Wenn Sie kein Aufhebungsdatum angeben, ist der Dauerauftrag unbefristet.	
Alle 1 Wochen immer am	Montag 🔍		
(1 = jede Woche)			
Ausführungsoptionen			
Bitte wählen Sie eine der folgenden Optionen für de Buchung am vorhergehenden Werktag ausfü	n Fall, dass die Ausführung Ihres E Ihren. • Buchung am nac	Dauerauftrags auf ein Wochenende oder einen Fe hfolgenden Werktag ausführen.	eiertag fällt:
Durch die Verschiebung kann es vorkommen, dass	der vorgesehene Monat verlassen ausführen. 🗌 Kostenpflichtig	wird. ge Buchungsbestätigung ?	

Abbildung 9.2: Anlegen eines Dauerauftrags am Beispiel der Postbank

## 9.1.1 Informationen vom Kreditinstitut holen

Um die für die Umsetzung einer entsprechenden Eingabemaske unter Access benötigten Informationen vom Kreditinstitut zu ermitteln, benötigen wir eine kleine Hilfsfunktion. Diese haben wir bereits in Kapitel »BACSegment-Anwendung: Die Bankparameterdaten (BPD)« ab Seite 107 vorgestellt. Die Funktion heißt *BPDSegment* und liefert den XML-Inhalt eines Segments des *BACBankData*-Objekts der mit *objContact* referenzierten Bank. Der Aufruf für das hier benötigte HIDAES-Segment sieht so aus:

```
Public Sub Test_BPDHIDAES()
   Dim objContact As BACContact
   Dim strXML As String
   Set objContact = GetContacts.Item(0)
   strXML = BPDSegment(objContact, eHIDAES)
   Debug.Print FormatXML(strXML)
End Sub
```

Für die Postbank beispielsweise sieht das so ermittelte XML-Dokument wie folgt aus:

```
<?xml version="1.0" encoding="UTF-16"?>
<SEGMENT TYPE="HICDES" VERSION="1" SEQNO="49" REFSEQNO="4">
    <MaxAnzahlAuftraege Format="numeric">1</MaxAnzahlAuftraege>
    <MinAnzahlSignaturen Format="numeric">1</MinAnzahlSignaturen>
    <Sicherheitsklasse Format="digits">1</Sicherheitsklasse>
    <Parameter>
        <MaxAnzahlVerwendungszweckzeilen Format="numeric">4
            </MaxAnzahlVerwendungszweckzeilen>
        <MinVorlaufzeit Format="numeric">1</MinVorlaufzeit>
        <MaxVorlaufzeit Format="numeric">180</MaxVorlaufzeit>
        <TurnusInMonaten Format="digits">00</TurnusInMonaten>
        <AusfuehrungstageProMonat Format="digits">00</AusfuehrungstageProMonat>
        <TurnusInWochen Format="digits">00</TurnusInWochen>
        <AusfuehrungstageProWoche Format="digits">12345
            </AusfuehrungstageProWoche>
    </Parameter>
</SEGMENT>
```

Dieses Dokument liefert uns die folgenden wichtigen Informationen:

- » MaxAnzahlAuftraege: Maximale Anzahl der anzulegenden Aufträge
- » MaxAnzahlVerwendungszweckzeilen: Maximale Anzahl der Verwendungszweckzeilen
- » MinVorlaufzeit und MaxVorlaufzeit: Anzahl der möglichen Tage vom aktuellen Tag bis zum Ausführungstag. Hier ist beispielsweise mindestens ein Tag Vorlaufzeit nötig, maximal sind 180 Tage erlaubt.
- » TurnusInMonaten: Mögliche Werte für den Turnus, hier alle. Es könnte auch ein Wert wie 0102030612 zurückgegeben werden, was bedeuten würde, dass alle 1, 2, 3, 6 oder 12 Monate als Turnus möglich sind.
- » AusfuehrungstageProMonat: Mögliche Werte für den Ausführungstag bei monatlicher Ausführung, hier alle Tage von 1 bis 30.
- » TurnusInWochen: Mögliche Werte für den wöchentlichen Turnus, hier 00 dies bedeutet alle
- » AusfuehrungstageProWoche: Tage der Woche, an denen der Dauerauftrag ausgeführt werden kann – hier 1, 2, 3, 4 und 5, also Montag bis Freitag.

Damit wissen wir also schon einmal, welche Datumsangaben wir dem Benutzer anbieten können, wenn er sich erst einmal für die Option *Woche* oder *Monat* entschieden hat. Das Formular zur Eingabe der Daten zum Anlegen eines Dauerauftrags sieht nun wie in Abbildung 9.3 aus.

📑 frmSEPADauerauftrag 📼 🗉	23
• • • • 1 • • 1 • • 2 • • • 3 • • • 4 • • • 5 • • • 6 • • • 7 • • • 8 • • • 9 • • • 10 • • • 11 • •	12 📥
✓ Formularkopf	
SEPA-Dauerauftrag	
Oetailbereich	_
Auftraggeber:	
Bankverbindung: Ungebunden	
- Konto/Depot: Ungebunden 💌	
2. IBAN: Ungebunden	
BIC: Ungebunden	
Zahlungsempfänger:	
Zahlungsempfänger: Ungebunden	
5 Bezeichnung: Ungebunden	
Emptangername 1: Ungebunden	
6 Empfangername 2: Ungebunden	
IBAN: Ungebunden	
7 Ungebunden	
- Sonstige Daten:	
Betrag: Ungebunden Mit Testdaten füllen	
Verwendungszweck: Ungebunden (noch 140 Zeichen)	
Ausführung alle ebun Wochen immer am Ungebunden	
12 Ausführung alle ebun Monate immer am ebun des Monats	
Erstmals ausführen: Ungebunden	
13 Letztmals ausführen: Ungebunden	
- 15	
✓ FormularfuB	
Einreichen 🔀 OK	-

Informationen für das Anlegen eines Dauerauftrags

Abbildung 9.3: Formular zur Eingabe der Daten eines Dauerauftrags

Den interessanten Teil haben wir farbig eingerahmt. Hier finden Sie die folgenden Steuerelemente:

» Optionsgruppe *ogrAusfuehrung*: hat keine Beschriftung und der Rahmen ist transparent dargestellt. Er nimmt die beiden Optionsfelder mit den Beschriftungen *Ausführung alle Wochen immer am und Ausführung alle … Monate immer am … des Monats* auf.

- » Über dem ersten Freiraum der oberen Option haben wir ein Kombinationsfeld namens cbo-Wochenturnus platziert, hinter der Beschriftung der oberen Option ein Kombinationsfeld namens cboAusfuehrungstagWoche. Das obere Optionsfeld erhält den Wert 1.
- » Der erste Freiraum in der Beschriftung des zweiten Optionsfeldes lässt Platz für das Textfeld cboMonatsturnus, der zweite für das Kombinationsfeld cboAusfuehrungstagMonat. Das Optionsfeld selbst erhält den Wert 2.
- » Darunter befindet sich das Kombinationsfeld cboErstmalsAusfuehrenAm.
- » Ganz unten legen Sie ein Kombinationsfeld namens cboLetztmalsAusfuehrenAm an.

Mit diesen Steuerelementen ausgestattet müssen wir nun festlegen, welche Auswirkungen Änderungen an den einzelnen Steuerelementen haben:

- » Wenn der Benutzer den Wert der Optionsgruppe ogrAusfuehrung ändert, müssen die Datensatzherkünfte und gegebenenfalls auch die Inhalte der beiden Kombinationsfelder cboErstmalsAusfuehrenAm und cboLetztmalsAusfuehrenAm angepasst werden. Letzteres, wenn dort bereits Werte ausgewählt waren und nicht mehr in der neu gefüllten Datensatzherkunft vorhanden sind.
- » Wenn der Benutzer den Wert der Kombinationsfelder cboAusfuehrungstagWoche oder cbo-AusfuehrungstagMonat ändert, müssen die Datensatzherkünfte der beiden Kombinationsfelder cboErstmalsAusfuehrenAm und LetztmalsAusfuehrenAm angepasst werden – so, dass diese nur noch die entsprechenden Tage anzeigen.
- » Wenn der Benutzer den Eintrag im Kombinationsfeld cboErstmalsAusfuehrenAm ändert, muss der Inhalt des Kombinationsfeldes cboLetztmalsAusfuehrenAm ebenfalls angepasst werden.

## 9.1.2 Dauerauftrag-geeignete Banken ermitteln

Im Gegensatz zu den gängigeren Vorgängen wie dem Abruf von Kontoständen oder dem Durchführen von SEPA-Lastschriften bietet nicht jede Bank die Möglichkeit an, Daueraufträge anzulegen. Wenn wir das obere Kombinationsfeld *cboContacts* mit den zur Verfügung stehenden Banken füllen, sollten wir also gleich diejenigen außen vor lassen, die dieses Feature nicht bieten.

Die Prozedur, die durch das Ereignis *Form\_Open* des Formulars ausgelöst wird, übernehmen wir in weiten Teilen von den bisher beschriebenen Formularen. Wir ändern allerdings eine Zeile, indem wir den Aufruf der Funktion *GetContactList* durch den der Funktion *GetContactListWithFeature* ersetzen (siehe fett markierte Zeile):

```
Private Sub Form_Open(Cancel As Integer)
Dim intLastContact As Integer
Dim intLastAccount As Integer
Me!cboContacts.RowSourceType = "Value List"
```

#### Informationen für das Anlegen eines Dauerauftrags

```
Me!cboAccounts.RowSourceType = "Value List"
   Me!cboContacts.RowSource = GetContactListWithFeature("HICDES")
   If len(Me!cboContacts.RowSource) = 0 Then
       MsgBox "Es wurden noch keine Kontakte angelegt." & vbCrLf & vbCrLf
           & "Dies können Sie über den Eintrag 'Homebanking Kontakte' in der "
           & "Systemsteuerung erledigen."
       Cancel = True
       Exit Sub
   End If
   intLastContact = Nz(DLookup("LetzterBankkontakt", "tbl0ptionen"), -1)
   If Not intLastContact = -1 Then
       Me!cboContacts = intLastContact ' Me!cboContacts.ItemData(intLastContact)
       If IsNull(Me!cboContacts.Column(0)) Then
           Me!cboContacts.SetFocus
           Me!cboContacts.Dropdown
       Else
           Me!cboAccounts.RowSource = GetAccountList(intLastContact, True)
           intLastAccount = Nz(DLookup("LetztesKonto", "tbl0ptionen"), -1)
           If Not intLastAccount = -1 Then
               Me!cboAccounts = Me!cboAccounts.ItemData(intLastAccount)
               Me!txtBICSender = Me!cboAccounts.Column(2)
               Me!txtIBANSender = Me!cboAccounts.Column(3)
           End If
       Fnd If
   Fnd If
End Sub
```

Diese Funktion erwartet als Parameter nicht nur den bereits bekannten und optionalen Wert für die *Boolean*-Variable *bolReset* (der dafür sorgt, dass die Kontakte neu eingelesen werden), sondern auch noch den Parameter *strSegment*. Diesem übergeben Sie in diesem Fall den Wert *HICDES*. *HICDES* ist das Segment, das von den Bankparameterdaten (*BPD*) geliefert wird und die Vorgaben für das Anlegen der Datumsangaben für den Dauerauftrag liefert. Sollte der aktuell als *intLastContact* festgelegte Kontakt nicht in der Liste der Kontakte auftauchen, die das Feature unterstützen, wird gar kein Kontakt ausgewählt.

Die Funktion *GetContactListWithFeature* ermittelt zunächst mit der bereits weiter oben beschriebenen Funktion *GetContacts* die aktuelle Liste der Kontakte aus der Systemsteuerung. Diese durchläuft sie in einer *For...Next*-Schleife. Dabei wird jeweils ein Eintrag der Auflistung *objContacts* mit der Variablen *objContact* referenziert. Für diesen prüft die Routine mit einer weiteren Funktion namens *BACBPDSegmentVorhanden*, ob das entsprechende Segment, hier *HICDES*, vorhanden ist. Falls ja, wird das *Contact*-Objekt zur Liste der Kontakte hinzugefügt:

Public Function GetContactListWithFeature(strSegment As String,

```
Optional bolReset As Boolean) As String
   Dim objContact As BACContact
   Dim strContacts As String
   Dim i As Integer
   Dim j As Integer
   Dim objContacts As BACContacts
   Set objContacts = GetContacts(bolReset)
   For i = 0 To objContacts.count - 1
       Set objContact = objContacts.Item(i)
       If BACBPDSegmentVorhanden(objContact, strSegment) Then
           strContacts = strContacts & i & ";" & objContact.UserID & ";"
               & objContact.BankCode & ";"
               & objContact("Contact")
               & "|" & objContact.BankCode
               & "|" & objContact.UserID & ";"
       End If
   Next i
   GetContactListWithFeature = strContacts
End Function
```

Die Funktion BACBPDSegmentVorhanden erwartet wiederum das BACContact-Element sowie den Namen des zu untersuchenden Segments als Parameter. Sie ermittelt zunächst mit der Funktion FindSegmentType der Segments-Auflistung des BankData-Objekts den Index für das Segment und speichert diesen in der Variablen intSegment. Hat intSegment danach den Wert -1, ist das Segment nicht vorhanden. In diesem Fall liefert BACBPDSegmentVorhanden den Wert False zurück, sonst den Wert True:

Damit zeigt das Kombinationsfeld *cboContacts* nun nur diejenigen Bankkontakte an, welche den Dauerauftrag anbieten.

Informationen für das Anlegen eines Dauerauftrags

## 9.1.3 Informationen für die Datumseingabe von der Bank einlesen

Nun lesen wir die Informationen ein, die von der Bank für die Zusammenstellung eines Dauerauftrags vorgegeben werden. Diese speichern wir in einigen Variablen, die wie folgt im Kopf des Klassenmoduls *frmSEPADauerauftrag* deklariert werden:

Dim intMinVorlaufzeit As Integer Dim intMaxVorlaufzeit As Integer Dim strTurnusInMonaten As String Dim strAusfuehrungstageProMonat As String Dim strTurnusInWochen As String Dim strAusfuehrungstageProWoche As String

#### Wozu dienen diese Variablen?

- » intMinVorlaufzeit: Gibt die minimale Vorlaufzeit in Tagen an.
- » intMaxVorlaufzeit: Gibt die maximale Vorlaufzeit in Tagen an.
- » strTurnusInMonaten: Gibt die möglichen Turnus an, zum Beispiel monatlich (01), zweimonatlich (02), quartalsweise (03), halbjährlich (06), jährlich (12).
- » strAusfuehrungstageProMonat: Legt die Ausführungstage fest. 00 bedeutet: Alle Ausführungstage sind möglich, also von 01 bis 30 (01020304...30). Wenn nicht alle Tage verwendet werden können, finden Sie einen String mit den Nummern der zulässigen Tage vor (01020304...30).
- » *strTurnusInWochen*: Gibt den Turnus in Wochen an. *00* bedeutet: Alle Turnus sind möglich. Wenn nur bestimmte Turnus möglich sind, nimmt diese Variable eine entsprechende Zeichenkette auf (zum Beispiel *010204*)
- » strAusfuehrungstageProWoche: Legt die Tage der Woche fest, an denen die Ausführung möglich ist. Die Zahl 1 entspricht dabei dem *Montag*, 2 heißt *Dienstag* uns so weiter. Wenn eine Ausführung nur von Montag bis Freitag möglich ist, erhält die Variable den Wert 12345.
- » Für das Einlesen dieser Parameter aus den Bankparameterdaten (*BPD*) haben wir eine eigene Prozedur erstellt, die wie folgt aussieht:

```
Private Sub ZeitparameterEinlesen()
Dim objContact As BACContact
Dim strXML As String
Set objContact = GetContacts.Item(Me!cboContacts)
strXML = BPDSegment(objContact, eHICDES)
intMinVorlaufzeit = GetXMLElement(strXML, "//SEGMENT/Parameter/MinVorlaufzeit")
intMaxVorlaufzeit = GetXMLElement(strXML, "//SEGMENT/Parameter/MaxVorlaufzeit")
strTurnusInMonaten = GetXMLElement(strXML, "//SEGMENT/Parameter/TurnusInMonaten")
```

```
strAusfuehrungstageProMonat = _
    GetXMLElement(strXML, "//SEGMENT/Parameter/AusfuehrungstageProMonat")
    strTurnusInWochen = GetXMLElement(strXML, "//SEGMENT/Parameter/TurnusInWochen")
    strAusfuehrungstageProWoche = GetXMLElement(strXML, "//SEGMENT/Parameter/
AusfuehrungstageProWoche")
End Sub
```

Die Prozedur ermittelt zunächst mit der Funktion *BPDSegment* das XML-Dokument des hier benötigten Segments mit der Kennung *HICDES*. Diese Funktion erwartet den Kontakt und die Konstante für das Segment (hier *eHICDES*) als Parameter und sieht wie folgt aus:

```
Public Function BPDSegment(objContact As BACContact, intSegment As eBPD) As String
  Dim objBankdata As BACBankData
  Dim objSegment As BACSegment
  Dim strSegment As String
  Set objBankdata = objContact.BankData
  strSegment = GetSegmentName(intSegment)
  intSegment = objBankdata.Segments.FindSegmentType(strSegment)
  Set objSegment = objBankdata.Segments(intSegment)
  BPDSegment = objSegment.GetXML
End Function
```

Die Funktion liest zunächst das *BankData*-Objekt des Kontakts ein und speichert diesen in der Variablen *objBankdata*. Dann ermittelt sie mit der weiter oben vorgestellten Funktion *GetSegmentName* den Namen des mit *intSegment* angegebenen Segments, hier *HICDES*.

Dann ermittelt sie mit der *FindSegmentType*-Methode für die Segmentkennung den Index des Segments und ermittelt damit das Segment aus der *Segments*-Auflistung des *BACBankData*-Objekts. Dieses liefert wiederum über die Eigenschaft *GetXML* das XML-Dokument mit den gesuchten Informationen.

Damit zurück zur Prozedur *ZeitparameterEinlesen*. Nachdem diese das XML-Dokument in die Variable *strXML* eingelesen hat, übergibt sie dieses für jeden zu ermittelnden Wert je einmal an die Funktion *GetXMLElement* (siehe »Die Funktion GetXMLElement«, Seite 447).

Auf diese Weise liest die Prozedur *ZeitparameterEinlesen* die sechs benötigten Informationen in die passenden Variablen ein.

Dies ist relativ viel Aufwand, um allein die Basisinformationen zum Vorbereiten der Steuerelemente für die Datumsangaben und deren Übergabe an die Bank zu realisieren, aber auf diese Weise halten Sie den Vorgang flexibel. Informationen für das Anlegen eines Dauerauftrags

### Zeitparameter beim Laden des Formulars einlesen

Zu welchem Zeitpunkt aber rufen wir die oben beschriebene Prozedur *Zeitparameter* eigentlich auf? Dies erledigen wir in der Prozedur, die durch das Ereignis *Beim Laden* des Formulars ausgelöst wird. Diese sieht wie folgt aus und ruft noch drei weitere Prozeduren auf:

```
Private Sub Form_Load()
    If Not IsNull(Me!cboContacts.Column(0)) Then
        ZeitparameterEinlesen
        BasisdatenAktualisieren
        ZeitraumdatenAktualisieren
    End If
End Sub
```

Der zweite Aufruf füllt die Datensatzherkünfte der vier Kombinationsfelder zur Auswahl der Voreinstellungen mit der Prozedur *BasisdatenAktualisieren*, der dritte füllt die Datensatzherkünfte der beiden Kombinationsfelder zur Auswahl des Zeitraums mit der Prozedur *Zeitraumdaten-Aktualisieren*. Beide Prozeduren stellen wir weiter unten vor.

Die *If...Then*-Bedingung prüft, ob aktuell ein Kontakt im Kombinationsfeld *cboContacts* ausgewählt ist und führt die drei Prozeduraufrufe nur in diesem Fall aus.

### Aktualisierung bei der Bankauswahl

Wenn der Benutzer mit dem Kombinationsfeld *cboContacts* eine andere als die beim Laden des Formulars eingestellte auswählt, müssen die Informationen für die entsprechende Bank eingelesen werden. Außerdem sollen auch die Steuerelemente zum Festlegen der Parameter für den Dauerauftrag aktualisiert werden. Deshalb statten wir die Prozedur, die durch das Ereignis *Nach Aktualisierung* des Kombinationsfeldes *cboContacts* ausgelöst wird, mit den gleichen drei Anweisungen aus, die schon beim Laden des Formulars aufgerufen werden:

```
Private Sub cboContacts_AfterUpdate()

If Not IsNull(Me!cboContacts) Then
Me!cboAccounts.RowSource = GetAccountList(Me!cboContacts, True)
Me!cboAccounts = Me!cboAccounts.ItemData(0)
Me!cboAccounts.Dropdown
ZeitparameterEinlesen
BasisdatenAktualisieren
Else
Me!cboAccounts.RowSource = ""
End If
End Sub
```

## 9.1.4 Basisdaten aktualisieren

Nachdem die Informationen aus den Bankparameterdaten in die sechs Variablen eingelesen wurden, können wir damit die Steuerelemente zum Definieren der Dauerauftragsparameter füllen – zunächst die vier Kombinationsfelder im oberen Bereich (siehe Abbildung 9.4).

E SEPA-Dauerauftrag			23
SEPA-Dauerauftrag			
Auftragg			-
<ul> <li>Ausführung alle</li> <li>Ausführung alle</li> <li>Monate immer am</li> <li>Mon</li></ul>	s Mona	] ats	
Einreichen 🔀 OK		6	3

Abbildung 9.4: Steuerelemente zur Eingabe der Dauerauftrag-Parameter

Dies erledigt die Prozedur *BasisdatenAktualisieren*. Diese füllt zunächst das Kombinationsfeld *cboMonatsturnus*. Wenn die Variable *strTurnusInMonaten* den Wert *00* aufweist, sollen alle Turnusse von 1 bis 24 im Kombinationsfeld auswählbar sein. Dazu stellt die Prozedur die Datensatzherkunft im ersten Teil der ersten *If...Then*-Bedingung auf den Ausdruck *1;2;3;...;24* ein. Damit diese Liste im Kombinationsfeld angezeigt wird, müssen Sie zuvor noch die Eigenschaft *Herkunftsart* auf den Wert *Wertliste* einstellen. Dies gilt übrigens für alle im Folgenden vorgestellten Kombinationsfelder.

Sollte *strTurnusInMonaten* hingegen selbst eine Liste möglicher Turnusse enthalten, sieht der Inhalt der Variablen beispielsweise so aus: *0102030612*. Dies bedeutet, dass die Turnusse ein Monat, zwei Monate, drei Monate, sechs Monate oder zwölf Monate angeboten werden sollen. Hier greift der zweite Teil der ersten *If...Then*-Bedingung. In einer *For...Next*-Schleife durchläuft die Prozedur hier alle Zeichen der Variablen *strTurnusInMonaten*, und zwar in Zweierschritten (*Step 2*). Dabei fügt sie die jeweiligen Werte zu einer in der Variablen *strRowSource* gespeicherten Datensatzherkunft zusammen, die dann beispielsweise so aussieht: *01;02;03;06;12* – damit ist das Kombinationsfeld *cboMonatsturnus* bereits gefüllt:

```
Private Sub BasisdatenAktualisieren()
Dim strRowsource As String
```

#### Informationen für das Anlegen eines Dauerauftrags

Damit es auch noch gleich beim Öffnen des Formulars beziehungsweise nach der Auswahl eines anderen Kontakts einen Wert anzeigt, stellt die Prozedur das Kombinationsfeld auf den ersten Wert ein:

```
Me!cboMonatsturnus = Me!cboMonatsturnus.ItemData(0)
```

Danach geht es mit dem Kombinationsfeld *cboWochenturnus* weiter. Hier untersucht die Prozedur die Variable *strTurnusInWochen*. Auch hier gilt: Der Wert *OO* steht stellvertretend für "keine Einschränkung". Die Prozedur füllt dann die Datensatzherkunft mit den Werten von 1 bis 24. Wenn *strTurnusInWochen* einen anderen Ausdruck enthält, durchläuft die Prozedur auch hier die Zeichenkette in Zweierschritten und fügt daraus die Datensatzherkunft des Kombinationsfeldes zusammen, bevor sie diese der entsprechenden Eigenschaft zuweist und auch hier den ersten Eintrag auswählt:

```
If strTurnusInWochen = "00" Then
    Me!cboWochenturnus.RowSource = "1:2;3:4;5:6;7:8;9;10;11:12;" _
    & "13:14:15:16:17:18:19:20:21:22:23:24"
Else
    strRowsource = ""
    For i = 1 To Len(strTurnusInWochen) Step 2
        strRowsource = strRowsource & Mid(strTurnusInWochen, i, 2) & ";"
    Next i
    Me!cboWochenturnus.RowSource = strRowsource
End If
Me!cboWochenturnus = Me!cboWochenturnus.ItemData(0)
```

Beim Kombinationsfeld *cboAusfuehrungstagMonat* sieht es ähnlich aus. Hier werden allerdings, wenn *strAusfuehrungstageProMonat* den Wert *00* aufweist, explizit die zur Verfügung stehenden Tage des Monats angezeigt – in diesem Fall die Tage von *1* bis *30*. Anderenfalls ermittelt die Prozedur wieder die Tage aus der Zeichenkette in *strAusfuehrungstageProMonat*:

```
If strAusfuehrungstageProMonat = "00" Then
```

Fehlen noch die Ausführungstage pro Woche, die das Kombinationsfeld *cboAusfuehrungstag-Woche* anbietet. Hier soll das Kombinationsfeld einen anderen Wert anzeigen, als es in der gebundenen Spalte enthält. Daher stellen Sie die Eigenschaft *Spaltenanzahl* auf den Wert 2 und *Spaltenbreiten* auf *Ocm* ein. Im Falle des Wertes 00 in der Variablen *strAusfuehrungstage* sollen alle Tage angeboten werden – also *0* für *Montag*, *1* für *Dienstag* und so weiter.

Enthält *strAusfuehrungstageProWoche* einen anderen Wert als *00*, handelt es sich beispielsweise um eine Zeichenkette wie *12345* (für die Tage *Montag* bis *Freitag*). Diese werden dann einfach in einer Schleife durchlaufen und in der Variablen *strRowSource* zusammengeführt. Dabei ist allerdings zu beachten, dass hier für jeden Wert erst der Zahlenwert und dann der mit *Format(Mid(strAusfuehrungstageProWoche, i, 1), "dddd")* ermittelte Name des Wochentags in *strRowSource* landen:

```
If strAusfuehrungstageProWoche = "00" Then
    Me!cboAusfuehrungstagWoche.RowSource = "0:'Montag':1:'Dienstag':2:'Mittwoch':" _
    & "3:'Donnerstag':4:'Freitag':5:'Samstag':6:'Sonntag'"
Else
    strRowsource = ""
    For i = 1 To Len(strAusfuehrungstageProWoche)
        strRowsource = strRowsource & Mid(strAusfuehrungstageProWoche. i. 1) _
            & ":" & Format(Mid(strAusfuehrungstageProWoche. i. 1) + 1. "dddd") & ":"
    Next i
    Me!cboAusfuehrungstagWoche.RowSource = strRowsource
End If
    Me!cboAusfuehrungstagWoche = Me!cboAusfuehrungstagWoche.ItemData(0)
End Sub
```

Nun haben wir für die vier oberen Kombinationsfelder die Datensatzherkunft gefüllt und diese jeweils auf den ersten verfügbaren Wert eingestellt. Außerdem hat die Optionsgruppe *ogrAus-fuehrung* standardmäßig bereits den Wert 1 erhalten, was zunächst die Daten für einen wöchentlichen Dauerauftrag liefert.

#### Informationen für das Anlegen eines Dauerauftrags

Die weiteren beiden Kombinationsfelder *cboErstmalsAusfuehrenAm* und *cboLetztmalsAusfuehrenAm* werden nun in Abhängigkeit der aktuellen Werte gefüllt. Dies erledigt die Prozedur *zeitraumdatenAktualisieren*, die auch noch zu verschiedenen anderen Gelegenheiten aufgerufen wird – die wir uns zunächst ansehen:

- » beim Laden des Formulars,
- » beim Auswählen eines neuen Kontakts mit dem Kombinationsfeld cboContacts,
- » beim Aktualisieren der Optionsgruppe ogrAusfuehrung und
- » beim Auswählen eines neuen Eintrags eines der Kombinationsfelder *cboAusfuehrungstag-Woche* und *cboAusfuehrungstagMonat*.

Die entsprechenden Prozeduren, die wir weiter oben noch nicht vorgestellt haben, sehen wie folgt aus:

```
Private Sub ogrAusfuehrung_AfterUpdate()
    ZeitraumdatenAktualisieren
End Sub
Private Sub cboAusfuehrungstagMonat_AfterUpdate()
    ZeitraumdatenAktualisieren
End Sub
Private Sub cboAusfuehrungstagWoche_AfterUpdate()
    ZeitraumdatenAktualisieren
End Sub
```

Die Prozedur *ZeitraumdatenAktualisieren* finden Sie hier, wieder in einzelne Abschnitte aufgeteilt. Zunächst der Prozedurkopf und die Deklarationszeilen:

```
Private Sub ZeitraumdatenAktualisieren()
   Dim datStart As Date
   Dim datAktuell As Date
   Dim intWochentagAktuell As Integer
   Dim strStart As String
   Dim strEnde As String
   Dim datEnde As Date
   Dim intMonat As Integer
   Dim i As Integer
   Dim datStartTemp As Date
```

Die Prozedur ermittelt zunächst das aktuelle Datum und speichert es in der Variablen *datAktuell*. Daraus leitet sie dann einen *Integer*-Wert ab, der den Wochentag repräsentiert – davon ausgehend, dass der Montag der Zahl 1 entspricht:

```
datAktuell = Date
intWochentagAktuell = Weekday(datAktuell, vbMonday) - 1
```

Danach unterscheidet die Prozedur nach dem Wert der Optionsgruppe *ogrAusfuehrung*. Der erste Teil behandelt den Wert 1 und somit den Fall des wöchentlichen Dauerauftrags:

```
Select Case Me!ogrAusfuehrung
Case 1 'Wochen
```

Hier wird für den Start zunächst das Datum ermittelt, das in der gleichen Woche wie das aktuelle Datum liegt, aber dem gewählten Ausführungswochentag entspricht:

datStart = datAktuell + Me!cboAusfuehrungstagWoche - 1 - intWochentagAktuell

Liegt der so ermittelte Wochentag in der Woche vor dem Tag, der dem aktuellen Datum entspricht, wird das Datum aus *datStart* um eine Woche in die Zukunft verschoben:

```
If intWochentagAktuell > Me!cboAusfuehrungstagWoche Then
    datStart = datStart + 7
End If
```

Nun kommt die Vorlaufzeit aus *intMinVorlaufzeit* ins Spiel. Wenn das Datum in *datStart* kleiner als das aktuelle Datum plus der in *intMinVorlaufzeit* gespeicherten Anzahl Tage ist, dann wird das Datum wiederum um eine Woche in die Zukunft verschoben – und zwar innerhalb einer Do While-Schleife und so lange, bis die Abbruchbedingung erfüllt ist:

```
Do While datStart < Date + intMinVorlaufzeit
    datStart = datStart + 7
Loop</pre>
```

Damit ist das erste mögliche Startdatum gefunden und wird zur *String*-Variablen *strStart* hinzugefügt, die später als Datensatzherkunft des Kombinationsfeldes *cboErstmalsAusfuehrenAm* dient:

```
strStart = strStart & datStart & ";"
```

Nun folgen die übrigen möglichen Startdaten. Warum noch mehr? Weil der Benutzer alle Startdaten zur Verfügung gestellt bekommen soll, die innerhalb des Intervalls liegen, das durch das aktuelle Datum und die beiden in *intMinVorlaufzeit* und *intMaxVorlaufzeit* gespeicherten Werte vorgegeben werden. Also starten wir eine *Do While*-Schleife, die so lange durchlaufen wird und weitere Startdaten an *strStart* anhängt, bis das in *datStart* gespeicherte Datum hinter dem aktuellen Datum plus der Anzahl Tage aus *intMaxVorlaufzeit* liegt:

```
i = 1
Do While datStart + 7 * i < Date + intMaxVorlaufzeit
strStart = strStart & datStart + 7 * i & ";"
i = i + 1</pre>
```

Leseprobe! Vollständiges Buch erhältlich unter www.amvshop.de im André Minhorst Verlag.

Informationen für das Anlegen eines Dauerauftrags

Loop

*strStart* sieht beim aktuellen Datum *4.6.2014*, *intMinVorlaufzeit* = 1, *intMaxVorlaufzeit* = 180 und dem Wochentag *Donnerstag* beispielsweise so aus:

05.06.2014;12.06.2014;19.06.2014;26.06.2014;03.07.2014;10.07.2014;...;27.11.2014;

Kommen wir nun zu den Monaten, die im zweiten Teil der *Select Case*-Bedingung bearbeitet werden. Hier prüft die Prozedur zuerst, ob das mit *DateSerial* aus dem Monat und dem Jahr des aktuellen Datums plus dem in *cboAusfuehrungstagMonat* angegebenen Tages zusammengesetzt Datum größer oder gleich dem aktuellen Datum plus dem Wert aus *intMinVorlaufzeit* ist. Beispiel: Aktuelles Datum *4.6.2014*, ermitteltes Datum bei *cboAusfuehrungstagMonat* = 1 ist der *1.6.2014*, dies ist also auf jeden Fall kleiner als das aktuelle Datum plus *intMinVorlaufzeit*. In diesem Fall wird *datStart* auf das Datum eingestellt, dass dem Tag aus *cboAusfuehrungstagMonat*, dem folgendem Monat und dem aktuellen Jahr zusammengesetzt wird. Wenn der Benutzer hier als Ausführungstag den *30*. gewählt hätte, wäre der *30.6.2014* als erstes Datum in *datStart* gelandet. Da es relativ kompliziert ist, den ersten möglichen Tag für einen solchen Dauerauftrag zu finden, haben wir dazu eine eigene Funktion geschrieben. Diese wird gleich zu Beginn der Zusammenstellung der möglichen Starttermine aufgerufen:

Die Funktion sieht wie folgt aus und erwartet das aktuelle Datum, die Mindestvorlaufzeit und den Ausführungstag als Parameter:

```
Public Function StartdatumDauerauftrag(datAktuell As Date. _
            intMinVorlaufzeit As Integer. intAusfuehrungstagMonat As Integer) As Date
   Dim datStartdatum As Date
   Dim i As Integer
   Do
        datStartdatum = DateSerial(Year(datAktuell), Month(datAktuell) + i, _
            intAusfuehrungstagMonat)
        Do While Month(datStartdatum) > Month(datAktuell) + i
            datStartdatum = datStartdatum - 1
        Loop
        i = i + 1
        Loop While Not datStartdatum >= datAktuell + intMinVorlaufzeit
        StartdatumDauerauftrag = datStartdatum
End Function
```

Die Prozedur durchläuft so lange eine *Do…While Loop*-Schleife, bis das in *datStartdatum* gespeicherte Datum nicht größer als das aktuelle Datum plus der minimalen Vorlaufzeit ist. Innerhalb dieser Schleife wird *datStartdatum* auf das dem aktuellen Jahr und dem aktuellen Monat plus

dem Tag aus *intAusfuehrungstagMonat* zusammengestellt, wobei der Monat mit jedem Schleifendurchlauf um *i* Monate erhöht wird – und *i* mit jedem Durchlauf um 1. Dies kann beispielsweise dann in die Hose gehen, wenn der Ausführungstag der 30. ist und der Starttag im *Februar* liegt, der ja nur maximal 28 oder 29 Tage hat. Dafür ist die innere *Do While*-Schleife gedacht: Diese prüft, ob der Monat *datStartdatum* größer als der Monat aus *datAktuell* ist, wobei zum Monat von *datAktuell* noch der Inhalt von *i* hinzuaddiert wird. Wenn *datStartdatum* also aus dem Monat und dem Jahr des aktuellen Datums (zum Beispiel 2/2014) und dem Tag aus *int-Ausfuehrungstag* zusammengesetzt wurde, hier 30, dann entsteht daraus nicht der 30.2.2014, sondern Access macht daraus den 2.3.2014. Der Monat dieses Datums (3) ist dann größer als der des Datums aus *datAktuell* (2), sodass so lange jeweils ein Tag von *datStartdatum* abgezogen wird, bis die Monate beider Datumsangaben übereinstimmen. Dies ist hier der Fall, wenn *datStartdatum* dem 28.2.2014 entspricht.

Danach wird das ermittelte Datum als erster Eintrag in die durch Semikola getrennte Liste der möglichen Startdaten eingetragen:

```
strStart = strStart & datStart & ";"
```

Schließlich sucht die Prozedur die weiteren Termine. Auch dazu verwenden wir eine eigene Funktion, die wie folgt aufgerufen wird:

```
strStart = strStart & WeitereStartdatenDauerauftrag(datStart, _
intMaxVorlaufzeit, Me!cboAusfuehrungstagMonat)
```

Die Funktion sieht schließlich wie folgt aus und erwartet das soeben ermittelte erste Datum für die Ausführung des Dauerauftrags, die maximale Vorlaufzeit sowie wiederum den Ausführungstag als Parameter:

```
Public Function WeitereStartdatenDauerauftrag(datAktuell As Date,
       intMaxVorlaufzeit As Integer, intAusfuehrungstagMonat As Integer) As String
   Dim datStartdatum As Date
   Dim strStart As String
   Dim intMonat As Integer
   Dim i As Integer
   Do
       datStartdatum = DateSerial(Year(datAktuell), Month(datAktuell) + i,
           intAusfuehrungstagMonat)
       If Month(datAktuell) + i > 12 Then
           intMonat = (Month(datAktuell) + i) Mod 12
           If intMonat = 0 Then intMonat = 12
       Else
           intMonat = Month(datAktuell) + i
       Fnd If
       Do While Month(datStartdatum) > intMonat
```

#### Informationen für das Anlegen eines Dauerauftrags

```
datStartdatum = datStartdatum - 1
Loop
If i > 0 Then
strStart = strStart & datStartdatum & ";"
End If
i = i + 1
Loop While Not datStartdatum >= datAktuell + intMaxVorlaufzeit
WeitereStartdatenDauerauftrag = strStart
End Function
```

Die Prozedur durchläuft wiederum eine *Do...While Loop*-Schleife, die diesmal endet, sobald das Datum aus *datStartdatum* größer als das Datum aus *datAktuell* plus der maximalen Vorlaufzeit ist. Die diesem Monat entsprechende Zahl speichern wir zuvor in einer Variablen namens *int-Monat*, da wir diese gegebenenfalls noch anpassen müssen – etwa, wenn ein Zeitraum von mehr als einem Jahr berücksichtigt werden soll. Innerhalb der Schleife ermittelt die Funktion den Tag des folgenden Monats, der den gewünschten Ausführungstag hat. Diese wird gegebenenfalls wieder verlegt, wenn der aktuelle Monat weniger Tage als in *intAusfuehrungstagMonat* angegeben hat. Alle gefundenen Datumsangaben werden in der Variablen *strStart* gespeichert, deren Ergebnis schließlich als Funktionswert zurückgegeben wird.

Zu guter Letzt schreibt die Prozedur die Datumsliste als Datensatzherkunft in das Kombinationsfeld *cboErstmalsAusfuehrenAm* und stellt den Inhalt dieses Steuerelements auf den ersten Eintrag ein. Danach ruft sie die Prozedur *LetztmalsAusfuehrenAmAktualisieren* auf, um den Inhalt des zweiten Kombinationsfeldes *cboLetztmalsAktualisierenAm* zu aktualisieren:

```
End Select
Me!cboErstmalsAusfuehrenAm.RowSource = strStart
Me!cboErstmalsAusfuehrenAm = Me!cboErstmalsAusfuehrenAm.ItemData(0)
LetztmalsAusfuehrenAmAktualisieren
End Sub
```

Diese Prozedur soll auch ausgeführt werden, wenn der Benutzer selbst den Wert des Kombinationsfeldes *cboErstmalsAusfuehrenAm* aktualisiert. Dies löst die folgende Prozedur aus:

```
Private Sub cboErstmalsAusfuehrenAm_AfterUpdate()
    LetztmalsAusfuehrenAmAktualisieren
End Sub
```

#### Die Prozedur LetztmalsAusfuehrenAktualisieren sieht so aus:

```
Private Sub LetztmalsAusfuehrenAmAktualisieren()
  Dim strEnde As String
  Dim datEnde As Date
  Dim datEndeTemp As Date
  Dim i As Integer
```

```
Select Case Me!ogrAusfuehrung
Case 1 'Wochen
datEnde = CDate(Me!cboErstmalsAusfuehrenAm) + 7 * CInt(Me!cboWochenturnus)
strEnde = strEnde & datEnde & ";"
For i = 1 To 100
strEnde = strEnde & datEnde + 7 * i * Me!cboWochenturnus & ";"
Next i
Case 2 'Monate
strEnde = WeitereStartdatenDauerauftrag(Me!cboErstmalsAusfuehrenAm, _
1000, Me!cboAusfuehrungstagMonat)
End Select
Me!cboLetztmalsAusfuehrenAm.RowSource = strEnde
Me!cboLetztmalsAusfuehrenAm = Me!cboLetztmalsAusfuehrenAm.ItemData(0)
End Sub
```

Die Prozedur behandelt wiederum zwei Fälle, die per *Select Case*-Bedingung aufgeteilt werden: die wöchentlichen und die monatlichen Daueraufträge. Bei den Wochen erfasst die Prozedur zunächst das erste Enddatum, ausgehend von dem im Kombinationsfeld *cboErstmalsAusfuehrenAm* ausgewählten Datum. Dieses liegt genau die Anzahl Wochen später, die dem Wert im Kombinationsfeld *cboWochenturnus* entspricht.

Dieses erste Datum landet gleich in der Variablen *strEnde*, welche später als Datensatzherkunft für das Kombinationsfeld *cboLetztmalsAusfuehrenAm* verwendet wird. In einer *For...Next*-Schleife ermittelt die Prozedur dann die Termine für 100 weitere Wochen (dies können Sie nach Wunsch ändern).

Im zweiten Teil der Select Case-Bedingung kümmert sich die Prozedur um das Zusammenstellen einer Liste monatlicher Termine, die zum Datum der ersten Ausführung und den übrigen Informationen passen. Glücklicherweise können wir hier die Funktion WeitereStartdatenDauerauftrag wiederverwenden. Dieser übergeben wir diesmal allerdings für den Parameter intMax-Vorlaufzeit einen wesentlich größeren Wert (hier 1.000 Tage), damit Termine für die nächsten gut drei Jahre angezeigt werden können.

Schließlich fügt die Prozedur die in *strEnde* gespeicherte Datensatzherkunft in die entsprechende Datenherkunft des Kombinationsfeldes *cboLetztmalsAusfuehrenAm* ein und wählt den ersten Eintrag des Kombinationsfeldes aus.

# 9.2 Dauerauftrag durchführen

Nachdem wir die Steuerelemente zum Hinzufügen der einzelnen Parameter für die Erstellung des Dauerauftrags erläutert haben, kümmern wir uns um die Durchführung des Dauerauftrags selbst. Dies geschieht in zwei Schritten:

- » Zunächst stellen wir die benötigten Informationen aus den Steuerelementen des Formulars zusammen.
- » Dann rufen wir eine Prozedur auf, der wir alle Informationen übergeben und die schließlich den Dauerauftrag anlegt.

## 9.2.1 Dauerauftrag vorbereiten

Wenn der Benutzer im Formular *frmSEPADauerauftrag* auf die Schaltfläche *cmdDauerauftrag* klickt, löst er die folgende Prozedur aus. Diese deklariert zunächst die benötigten Variablen:

```
Private Sub cmdDauerauftrag_Click()
Dim intContact As Integer
Dim strZahlungsempfaenger1 As String
Dim strZahlungsempfaenger2 As String
Dim strZahlungsempfaengerIBAN As String
Dim strZahlungsempfaengerBIC As String
Dim strErstmalsAusfuehrenAm As String
Dim strLetztmalsAusfuehrenAm As String
Dim strTurnus As String
Dim strTurnus As String
Dim strZahlungstag As String
Dim strZeiteinheit As String
Dim objKontakt As BACContact
Dim objZahlungsempfaenger As BACAccount
Dim objAuftraggeber As BACAccount
```

Danach folgt die übliche Prüfung, ob die Daten des Zahlungsempfängers in der Tabelle *tblEmp-faenger* gespeichert wurde und, falls dies nicht der Fall ist, die Rückfrage, ob der Dauerauftrag dennoch eingereicht werden soll:

Die Variable *intContact* nimmt den Index des aktuell ausgewählten Bankkontakts auf (diese entspricht auch dem Index, unter dem die Kontakte in dem über die Systemsteuerung erreichbaren Verwaltungstool gespeichert sind): Leseprobe! Vollständiges Buch erhältlich unter www.amvshop.de im André Minhorst Verlag.

#### Kapitel 9 SEPA-Daueraufträge

intContact = Me!cboContacts

Einige Variablen werden dann mit den Werten der entsprechenden Textfelder gefüllt:

strZahlungsempfaenger1 = Me!txtName1 strZahlungsempfaenger2 = Nz(Me!txtName2) strZahlungsempfaengerIBAN = Me!txtIBANEmpfaenger strZahlungsempfaengerBIC = Me!txtBICEmpfaenger

Nun speichert die Prozedur einen Verweis auf das *BACContact*-Objekt des gewählten Kontakts in der Variablen *objKontakt*:

```
Set objKontakt = GetContacts.Item(Me!cboContacts)
```

Als Auftraggeber wird das mit dem Kombinationsfeld *cboAccounts* ausgewählte Konto erfasst und als Objekt des Datentyps *BACAccount* in der Variablen *objAuftraggeber* gespeichert:

Set objAuftraggeber = objKontakt.Accounts(Me!cboAccounts)

Schließlich fehlt noch ein *BACAccount*-Objekt für den Zahlungsempfänger. Dieses erstellen wir mit der Funktion *AccountErstellen*, dass in Kapitel »SEPA-Überweisung« ab Seite 251 genau erläutert wird.

```
Set objZahlungsempfaenger = AccountErstellen(strZahlungsempfaenger1, _
strZahlungsempfaenger2, strZahlungsempfaengerBAN, strZahlungsempfaengerBIC)
```

Die Ausführungsdaten für die erste und die letzte Ausführung schreiben wir zunächst im entsprechenden Format in die Variablen *strErstmalsAusfuehrenAm* und *strLetztmalsAusfuehrenAm*:

```
strErstmalsAusfuehrenAm = Format(Me!cboErstmalsAusfuehrenAm, "yyyymmdd")
strLetztmalsAusfuehrenAm = Format(Me!cboLetztmalsAusfuehrenAm, "yyyymmdd")
```

Nun folgt die obligatorische Fallunterscheidung nach wöchentlichem oder monatlichem Dauerauftrag, die wie gewohnt per *Select Case*-Bedingung aufgeteilt wird. Bei der Woche stellen wir den Turnus in der Variablen *strTurnus* ein, speichern den Ausführungstag und legen für die Variable *strZeiteinheit* den Wert *W* für *Woche* fest. Beim Monat brauchen wir den Monatsturnus sowie den Buchstaben *M* wie *Monat*:

```
If Me!ogrAusfuehrung = 1 Then 'Woche
    strTurnus = Me!cboWochenturnus
    strAusfuehrungstag = Me!cboAusfuehrungstagWoche
    strZeiteinheit = "W"
Else 'Monat
    strTurnus = Me!cboMonatsturnus
    strAusfuehrungstag = Me!cboAusfuehrungstagMonat
    strZeiteinheit = "M"
End If
```

Leseprobe! Vollständiges Buch erhältlich unter www.amvshop.de im André Minhorst Verlag.

#### Dauerauftrag durchführen

Schließlich ruft die Prozedur die Funktion *SEPADauerauftragErstellen* auf, wobei die oben zusammengestellten Informationen als Werte für die Parameter übergeben werden:

Das war es – wir gehen nun zur Beschreibung der Funktion SEPADauerauftragErstellen über.

## 9.2.2 Dauerauftrag durchführen

Diese Funktion ist im Modul *mdIOBMA\_SEPADauerauftraege* zu finden und erwartet die folgenden Parameter:

- » objContact: BACContact-Objekt mit dem Bankkontakt des ausführenden Kreditinstituts
- » objAuftraggeber: BACAccount-Objekt für den Auftraggeber
- » objEmpfaenger: BACAccount-Objekt für das Empfängerkonto
- » curBetrag: Zu überweisender Betrag
- » strVerwendungszweck: Verwendungszweck
- » strErstmalsAusfuehrenAm: Erstes Ausführungsdatum
- » strLetztmalsAusfuehrenAm: Letztes Ausführungsdatum
- » strTurnus: Turnus für die Anzahl der Wochen beziehungsweise Monate zwischen zwei Überweisungen des Dauerauftrags
- » strAusfuehrungstag: Bei wöchentlicher Überweisung Nummer des Tages (1 bis 7) oder bei monatlicher Überweisung der Tag des Datums (1 bis 31)
- » strZeiteinheit: W für Woche oder M für Monat

#### Die Prozedur beginnt wie folgt:

# 10 Lastschriften mit SEPA

Lastschriften waren bereits vor SEPA ein sehr interessantes Thema in Zusammenhang mit Datenbankanwendungen. Neben der herkömmlichen Rechnung ist ja gerade die Lastschrift ein für beide Seiten bequemes Werkzeug des bargeldlosen Zahlungsverkehrs. Im Gegensatz zur Überweisung, die vom Zahlungspflichtigen ausgelöst wird, erteilt der Zahlungspflichtige bei der Lastschrift dem Zahlungsempfänger das Recht, die Zahlung beziehungsweise die Abbuchung auszulösen.

Wenn man die Bank des Zahlungsempfängers als Inkassostelle bezeichnet und die Bank des Zahlungspflichtigen als Zahlstelle, dann erteilt der Zahlungsempfänger der Inkassostelle den Auftrag, die Lastschrift einzuziehen.

Mit der Einführung von SEPA wurden zwei europäische Lastschriftverfahren entwickelt:

- » die Basis-Lastschrift und
- » die Firmen-Lastschrift.

Während Lastschriften früher per Datenträgeraustauschformat eingereicht werden konnten, gibt es nun mit SEPA das *pain008*-Format.

Auch die DDBAC-Komponenten unterstützen SEPA-Lastschriften. Dies hat den Vorteil, dass Sie sich um Spezifikationen wie etwa des *pain008*-Formats nicht kümmern müssen – Sie können die Daten der Lastschrift einfach über das Objektmodell eingeben.

# 10.1 Grundlagen zu Lastschriften

Die SEPA-Lastschrift löst die bekannte Lastschrift ab. Dabei gibt es unter SEPA zwei Verfahren:

- » Die SEPA-Basislastschrift (SEPA Core Direct Debit) und
- » die SEPA-Firmenlastschrift (SEPA Business to Business Direct Debit).

Zwischen diesen beiden SEPA-Lastschrifttypen gibt es ein paar Unterschiede. Schauen wir uns zunächst die SEPA-Basislastschrift an und dann die SEPA-Firmenlastschrift.

# 10.1.1 SEPA-Basislastschrift

Die SEPA-Basislastschrift zeichnet sich durch die folgenden Merkmale aus:

- » Ein SEPA-Lastschriftmandat ist erforderlich.
- » Im Original vorliegende Einzugsermächtigungen können umgewidmet werden.

Kapitel 10 Lastschriften mit SEPA

- » Die Lastschrift kann innerhalb von acht Wochen nach Belastung "wegen Widerspruch" zurückgegeben werden.
- » Nicht autorisierte Lastschriften, etwa wegen eines fehlerhaften (nicht unterschriebenen) Mandats, können innerhalb von 13 Monaten zurückgegeben werden.
- » 14 Tage vor dem Einzug erfolgt, soweit nicht anders vereinbart, eine Vorabinformation (Pre-Notification). Auf diese Weise kann der Zahlungspflichtige für entsprechende Deckung des Kontos sorgen.
- » Die SEPA-Basislastschrift erfordert die Angabe des Betrages, der genauen Fälligkeit, der Mandatsrefenz und der Gläubiger-ID.

Die SEPA-Basislastschrift steht sowohl Verbrauchern als auch Unternehmern zur Verfügung.

## 10.1.2 SEPA-Firmenlastschrift

Die SEPA-Firmenlastschrift hat folgende Eigenschaften:

- » Für die SEPA-Lastschrift ist ebenfalls ein Mandat erforderlich.
- » Bestehende Abbuchungsaufträge können nicht umgewidmet werden.
- » Sie kann nur zu Lasten von Firmen gezogen werden.
- » Der Bank des Zahlungsempfängers muss ein Original des Mandats vorliegen, das dieser direkt vom Zahlungspflichtigen zugesendet wurde.
- » Die Lastschrift kann "wegen Widerspruch" nur vor dem Fälligkeitsdatum durch Widerruf zurückgegeben werden.
- » Auch hier erfolgt 14 Tage vor dem Einzug eine Vorabinformation, sofern nicht anders vereinbart.
- » Ebenso müssen hier der Betrag, die genaue Fälligkeit, die Mandatsrefenz und die Gläubiger-ID angegeben werden.

Achtung: immer zwei Originale mit Unterschriften anfertigen

# 10.1.3 International

Grenzüberschreitende Lastschriften im EURO-Zahlungsraum werden erst mit der SEPA-Lastschrift möglich. Dementsprechend verwenden Sie bei SEPA-Lastschriften die IBAN (International Bank Account Number, internationale Kontonummer) und BIC (Business Identifier Code, internationale Bankleitzahl) anstatt Kontonummer und Bankleitzahl.

Grundlagen zu Lastschriften

## 10.1.4 Schlüsselwörter

Nachfolgend einige wichtige Schlüsselwörter, die Sie in diesem Kapitel noch häufiger antreffen werden:

Jeder Lastschrifteinreicher (Zahlungsempfänger) besitzt eine individuelle Kennung zur Identifizierung, die sogenannte Gläubiger-Identifikationsnummer (CI, Creditor Identifier). Diese Nummer und die vom Zahlungsempfänger jedem Mandat zuzuordnende Mandatsreferenz (z.B. Rechnungsnummer) ermöglichen dem Zahler einen einfachen Abgleich von Belastungen auf seinem Zahlungskonto / Girokonto. Die Gläubiger-Identifikationsnummer ist in Deutschland 18 Stellen lang und wird von der Deutschen Bundesbank vergeben.

- » GläubigerID (CI, Creditor Identifier): Ist eine 18-stellige Zahl. Muss jede natürliche oder juristische Person bei der Deutschen Bundesbank beantragen. Sie muss dem Zahlungspflichtigen vor der ersten Lastschrift (siehe Pre-Notifcation) sowie mit jeder Lastschrift (im SEPA-XML) mitgeteilt werden.
- » Mandantsreferenz: wird im Mandat zwischen Gläubiger und Zahlungspflichtigem vereinbart. Gemeinsam mit der GläubigerID ist die Mandatsreferenz eindeutig (es darf also keine zwei Mandate eines Gläubigers mit der gleichen Mandatsreferenz geben). Die Mandatsreferenz ist mit der GläuberID mit der Pre-Notification und jeder SEPA-Lastschrift mitzuteilen.
- » Pre-Notification (Vorabinformation): Der Zahlungspflichtige muss frühzeitig (in der Regel 14 Tage vor Einreichung der Lastschrift) informiert werden. Neben Gläubiger-ID und Mandatsreferenz muss die Pre-Notification auch das Lastschriftdatum und den genauen Betrag beinhalten. Es können mehrere Vorabinformationen zusammengefasst werden (zum Beispiel Vorabinformation über die nächsten zwölf Lastschriften bei monatlicher Belastung)
- » SEPA-Lastschriften müssen terminiert eingereicht werden. Terminiert auf fünf Bankarbeitstage in der Zukunft bei erstmaliger Lastschrift, zwei Bankarbeitstage in die Zukunft bei Folgelastschrift und ein Bankarbeitstag in die Zukunft bei bestimmten Lastschriften (mit verkürzter Vorlauffrist – in der Regel nur für Zahlungsdienstleister).
- » Das Mandat ersetzt die bisherige Einzugsermächtigung.

## 10.1.5 Verschiedene Lastschriftarten

Für die beiden Lastschriftarten gibt es entsprechende Schlüsselwörter, die später in der Formulierung der XML-Dokumente zum Durchführen der Lastschrift auftauchen:

- » Basislastschrift: Kürzel CORE
- » Firmenlastschrift: Kürzel B2B

Kapitel 10 Lastschriften mit SEPA

Außerdem gibt es einzelne Lastschriften und Lastschriftfolgen. Bei einer Lastschriftfolge gibt es eine erste, eine oder mehrere folgende und eine letztmalige Lastschrift. Für die verschiedenen Typen gibt es wiederum Kürzel:

- » Einmal-Lastschrift: OOFF
- » Erst-Lastschrift: FRST
- » Folge-Lastschrift: RCUR
- » Letztmalige Lastschrift: FNAL

## 10.1.6 Einlieferung der Lastschrift

Sie geben immer ein Fälligkeitsdatum für eine Lastschrift an. Damit dieses eingehalten werden kann, müssen Sie bestimmte Fristen einhalten. Nachfolgend sehen Sie die Fristen für die verschiedenen Kombinationen:

- » Basislastschrift, Erst- und Einmal-Lastschrift: frühestens 14 Kalendertage vor der Fälligkeit und spätestens fünf *TARGET2*-Geschäftstage vor der Fälligkeit
- » Basislastschrift, Folge- und letztmalige Lastschrift: frühestens 14 Kalendertage vor der Fälligkeit und spätestens zwei TARGET2-Geschäftstage vor der Fälligkeit
- » Firmenlastschrift: frühestens 14 Tage vor der Fälligkeit und spätestens einen TARGET2-Geschäftstag vor der Fälligkeit

TARGET2-Geschäftstage sind alle Tage außer den folgenden Feiertagen:

- » Neujahr
- » Karfreitag
- » Ostermontag
- » Maifeiertag
- » 1. Weihnachtstag
- » 2. Weihnachtstag

## 10.1.7 SEPA-Lastschriftmandat

Wenn Sie SEPA-Lastschriften einreichen möchten, benötigen Sie ein Mandat vom Zahlungspflichtigen. Dieses SEPA-Lastschriftmandat ist der Nachfolger der Einzugsermächtigung.

Ein Dokument auf der Internetseite von *Die Deutsche Kreditwirtschaft* beschreibt umfassend die verschiedenen Dokumente, mit denen Sie ein SEPA-Lastschriftmandat einholen können:

#### Grundlagen zu Lastschriften

http://www.die-deutsche-kreditwirtschaft.de/uploads/media/120720\_DK\_Beispiele\_Muster\_ SEPA Lastschriftmandat-SDD Basis-Core 09072012.pdf

Das Dokument bezieht sich explizit auf deutschsprachige Zahler mit einem Konto in Deutschland. Es zeigt, wie Sie das Lastschriftmandat mit einem separaten Formular also auch mit einer Kombination aus Vertrag und Lastschriftmandat einholen. Nachfolgend die wichtigsten Informationen aus diesem Dokument.

## Inhalt des Lastschriftmandats

Das Lastschriftmandat sollte einen Text wie den folgenden beinhalten:

Ich ermächtige (Wir ermächtigen) [Name des Zahlungsempfängers], Zahlungen von meinem (unserem) Konto mittels Lastschrift einzuziehen. Zugleich weise ich mein (weisen wir unser) Kreditinstitut an, die von [Name des Zahlungsempfängers] auf mein (unser) Konto gezogenen Lastschriften einzulösen.

Hinweis: Ich kann (Wir können) innerhalb von acht Wochen, beginnend mit dem Belastungsdatum, die Erstattung des belasteten Betrages verlangen. Es gelten dabei die mit meinem (unserem) Kreditinstitut vereinbarten Bedingungen.

Außerdem muss das Mandat die folgenden Informationen enthalten:

- » Name, Adresse und Gläubiger-Identifikationsnummer.
- » Mandatsreferenz
- » Angabe, ob das Mandat f
  ür wiederkehrende Zahlungen oder eine einmalige Zahlung gegeben wird.
- » Name, Adresse, Kontoverbindung und Unterschrift des Kontoinhabers sowie Datum der Unterschrift.

## Gläubiger-Identifikationsnummer

Diese Nummer müssen Sie beantragen, wenn Sie SEPA-Lastschriften einreichen möchten. Dies erledigen Sie nicht etwa bei Ihrer Bank, sondern bei der Deutschen Bundesbank. Dort starten Sie mit dem folgenden Link:

https://extranet.bundesbank.de/scp/

Dort erwartet Sie zunächst einiges zu lesen, bevor Sie den Inhalt akzeptieren und per Klick auf die Schaltfläche *Weiter* zum nächsten Schritt vorangehen. Hier wählen Sie zunächst aus, zu welcher Personengruppe der Gläubiger gehört (siehe Abbildung 10.1).

Kapitel 10 Lastschriften mit SEPA

H Deutsche Bundesbank SEF ×		X
← → C 🔒 https://extranet.bundesbank.de/scp/beantragung_prep.do	5	Ξ
Gläubiger-Identifikationsnummer		<b>^</b>
Startseite		
		- 11
Beantragung der Gläubiger-Identifikationsnummer		
Bitte wählen Sie eine der unten aufgelisteten Personengruppen, für die die Gläubiger-Identifikationsnummer I werden soll.	eantragt	
Liste der Personengruppen		
<ul> <li>Natürliche Personen und Einzelunternehmen, Freiberufler</li> <li>Personenvereinigungen</li> <li>Juristische Personen des Privatrechts (z. B. AG, GmbH, e.V.)</li> <li>Juristische Personen des öffentlichen Rechts</li> </ul>		
Weiter	p=1-10-10-10-10-10	

Abbildung 10.1: Auswahl einer Personengruppe

Im zweiten Schritt legen Sie dann die Rechtsform für den Gläubiger fest (siehe Abbildung 10.2).

H Deutsche Bundesbank SEF ×		x
← → C 🔒 https://extranet.bundesbank.de/scp/eingabe_rechtsform.do	52	≡
		•
Startseite		
Beantragung der Gläubiger-Identifikationsnummer		
Bitte wählen Sie eine der unten aufgelisteten Rechtsformen.		
Liste der Rechtsformen		
● GbR		
<ul> <li>Verein (nicht eingetragen)</li> </ul>		
<ul> <li>Partnerschaft</li> </ul>		
O HG		
⊖ KG		
GmbH & Co. KG		
EWIV     Constitute Decomposition		
<ul> <li>Sonstige Personenvereinigung</li> </ul>		
Weiter		

Abbildung 10.2: Auswahl der Rechtsform

Grundlagen zu Lastschriften

Schließlich geben Sie die Daten des Antragstellers ein (siehe Abbildung 10.3).

Heutsche Bundesbank SEF ×	
← → C 🗋 https://extranet.bundesbank.de/scp/	/eingabe_daten.do 없 :
DEUTSCHE BUNDESBANK EUROSYSTEM Gläubiger-lo	dentifikationsnummer
Startseite	
Beantragung der Gl	äubiger-Identifikationsnummer
Bitte geben Sie Ihre Daten in die unten stehenden Feld Bundesrepublik Deutschland zulässig sind.	ler ein. Dabei ist zu beachten, dass nur Anschriften innerhalb der
Die mit * gekennzeichneten Felder sind Pflichtfelder.	
Daten d	des Antragstellers
Name oder Firma ^	
PLZ / Ort *	
Daten zu	ur Ansprechperson
Bei den angeforderten Daten zur Ansprechperson gebe Adresse der Person an, die die Antragstellung bearbeit	n Sie bitte den Namen, die Telefonnummer und die E-Mail- tet.
Anrede *	Frau 🔻
Name *	
Vorname *	
E-Mail-Adresse *	
Wiederholung E-Mail-Adresse *	
	Weiter
< <zurück< th=""><th></th></zurück<>	

Abbildung 10.3: Eingabe der Daten des Antragstellers

Anschließend haben Sie Gelegenheit, die eingegebenen Daten zu prüfen und können diese dann absenden. Nachdem Sie den Antrag auf diese Weise gestellt haben, erhalten Sie eine E-Mail mit einem Bestätigungslink. Bestätigen Sie diesen, erhalten Sie eine weitere E-Mail mit einem PDF-Dokument, das schließlich die Gläubiger-ID enthält (siehe Abbildung 10.4). Kapitel 10 Lastschriften mit SEPA



Abbildung 10.4: Dokument mit der Gläubiger-ID

## Mandatsreferenz

Die Mandatsreferenz liefert zusammen mit der Gläubiger-Identifikationsnummer eine eindeutige Bezeichnung für das jeweilige Mandat. Sie ist bis zu 35 alphanumerische Stellen lang und kann entweder bereits im Mandat enthalten sein oder dem Zahlungspflichtigen nachträglich bekanntgegeben werden.

## 10.1.8 Benötigte Daten

Wenn Sie eine Lastschrift einreichen möchten, müssen Sie zusätzlich zu den üblichen Daten der beiden beteiligten Parteien (Auftraggeber und Zahlungspflichtiger) also noch drei weitere Informationen zur Hand haben:

- » Gläubiger-ID
- » Mandatsreferenz
- » Datum der Mandatsreferenz

# 10.1.9 Nötige Informationen im Überblick

Bevor wir uns mit der Programmierung der verschiedenen Varianten von Lastschriften befassen, schauen wir uns diese noch einmal im Überblick an – und wo sollten wir dies besser erledigen können als mit dem entsprechenden Formular des Onlineportals einer Bank? In diesem Fall handelt es sich um eine Lastschrift, die für ein Konto der Commerzbank angelegt werden soll.

Abbildung 10.5 zeigt das entsprechende Onlineformular. Hier wählen Sie zunächst das Auftraggeberkonto aus, sofern mehrere verfügbar sind. Dann geben Sie an, ob es sich um eine Basislastschrift oder eine Firmenlastschrift handelt. Der Wechsel zwischen den beiden Optionen zeigt: Es gibt tatsächlich keine Unterschiede bezüglich der notwendigen einzugebenden Daten. Lediglich die Voraussetzungen unterscheiden sich – für die Basislastschrift reicht es, wenn der Zahlungspflichtige Ihnen ein Mandat erteilt, bei Firmen muss der Zahlungspflichtige seine Bank beauftragen, dem Empfänger den Einzug von Lastschriften zu gestatten. Davon abgesehen gelten unterschiedliche Regeln bezüglich des Ausführungstermins, die wiederum vom Sequenztyp abhängig sind (siehe weiter oben).

Den Sequenztyp kann man etwa bei der Commerzbank per Auswahlfeld selektieren – hier stehen die Werte *Einmal-Lastschrift (OOFF), Erst-Lastschrift (FRST), Folge-Lastschrift (RCUR)* und *Letztmalige Lastschrift (FNAL)* zur Verfügung.

Danach geben Sie im Formular der Commerzbank den Namen (70 Zeichen), die IBAN und die BIC des Zahlungspflichtigen ein, außerdem den einzuziehenden Betrag. Der Verwendungszweck kann, aber muss nicht ausgefüllt werden (140 Zeichen).

Die folgenden drei Werte sind die Mandatsreferenz (etwa Kundennummer und/oder Rechnungsnummer), das Datum er Erteilung des Mandats und die Gläubiger-ID des Zahlungsempfängers. Kapitel 10 Lastschriften mit SEPA

Auftraggeberkonto	411890700   EUR   BLZ 35040038   ANDRE MINHORST
Lastschriftart	Basislastschrift     O Firmenlastschrift
Sequenz	Bitte treffen Sie eine Auswahl 🔻 i
Zahlungspflichtiger	
IBAN	i
BIC	<i>i</i> BIC suchen
Kreditinstitut	(automatisch)
Betrag	EUR
Verwendungszweck	
	140 Zeichen stehen noch zur Verfügung
Mandatsreferenz	i
Unterschriftsdatum	iii i
Gläubiger-ID	i
Ausführungstermin	nächstmögliche Ausführung 🔹 👔
Vorlage speichern	Ja, unter dem Namen

Abbildung 10.5: Formular zur Eingabe der Informationen zu einer Lastschrift

# 10.2 Basis-Lastschriften durchführen

Im ersten Beispiel schauen wir uns an, wie Sie eine einfache Basis-Lastschrift durchführen. Das Formular für diesen Geschäftsvorfall sieht wie in Abbildung 10.6 aus.

Es enthält neben den bereits bekannten Feldern noch drei weitere Felder:

- » txtGlaeubigerID: Identifikationsnummer des Gläubigers
- » txtMandatsreferenz: Mandatsreferenz
- » txtMandatsdatum: Datum, an dem das Mandat erstellt wurde

Um diese und einige weitere Felder schnell zu Testzwecken zu füllen, haben wir eine kleine Prozedur angelegt, die durch die Schaltfläche rechts neben dem Formular ausgeführt wird:

```
Private Sub cmdMitTestdatenFuellen_Click()
    Me!txtBetrag = 1
```

Basis-Lastschriften durchführen

```
Me!txtVerwendungszweck = "Testlastschrift"
Me!txtAusfuehrungsdatum = Date + 14
Me!txtGlaeubigerID = "DE98ZZZ09999999999"
Me!txtMandatsdatum = CDate("1.1.2014")
Me!txtMandatsreferenz = "AA00CreditorId"
```

End Sub



Abbildung 10.6: Formular für eine einfache Lastschrift im Entwurf

Kapitel 10 Lastschriften mit SEPA

Wenn Sie nun in die Formularansicht wechseln, wählen Sie ganz oben den Auftraggeber (aus Ihren eigenen Bankkonten, die Sie über die Systemsteuerung eingegeben haben) aus. Dann folgt der Zahlungspflichtige, den Sie mit dem Kombinationsfeld im zweiten Bereich auswählen oder manuell eingeben.

Schließlich fehlen noch die für die Lastschrift relevanten Daten, die Sie in diesem Formular manuell eingeben müssen. Zu Testzwecken füllen wir diese Felder schnell mit einem Mausklick auf die Schaltfläche *Mit Testdaten füllen*.

😑 Überweisung			23
SEPA-	Einzellastschrift		
Auftraggeber:			
Bankverbindung:	Datadesign Demobank 160 70000997 73516	0	-
Konto/Depot:	1000735160 Girokonto		•
IBAN:	DDBADEMM002		
BIC:	DE43700009971000735160		
Zahlungspflichtiger:			
Zahlungspflichtiger:	DataDesign Demobank 735161		2
Bezeichnung:	DataDesign Demobank 735161		
Empfängername 1:	DataDesign Demobank 75		
Empfängername 2:			
IBAN:	DE16700009971000735161		
BIC:	DDBADEMM002		
Sonstige Daten:			
Betrag:	1		
Verwendungszweck: (noch 140 Zeichen)	Testlastschrift		
Ausführungsdatum:	21.05.2014		
Gläubiger-ID:	DE98ZZZ09999999		
Mandatsrefererenz:	AA00CreditorId füllen		
Mandatsdatum:	01.01.2014		
Meldung:			
🔠 Einreichen	Х ок		

Abbildung 10.7: Das Formular frmSEPAEinzellastschrift mit Beispieldaten

# **10.2.1** Lastschrift starten

Ein Klick auf die Schaltfläche *cmdLastschrift* löst nun die folgende Ereignisprozedur aus:
Basis-Lastschriften durchführen

```
Private Sub cmdLastschrift Click()
   Dim intContact As Integer
   Dim strZahlungspflichtiger1 As String
   Dim strZahlungspflichtiger2 As String
   Dim strZahlungspflichtigerIBAN As String
   Dim strZahlungspflichtigerBIC As String
   Dim objKontakt As BACContact
   Dim objZahlungspflichtiger As BACAccount
   Dim objAuftraggeber As BACAccount
   Dim lngAuftragID As Long
   If EmpfaengerSpeichern = False Then
        If MsgBox("Die Empfängerdaten wurden nicht gespeichert. Überweisung dennoch "
                & "durchführen?", vbYesNo + vbExclamation,
                "Empfängerdaten nicht gespeichert") = vbNo Then
           MsgBox "Die Überweisung wurde abgebrochen.", vbCritical + vbOKOnly,
                "Vorgang abgebrochen"
           Exit Sub
       Fnd If
   Fnd If
    intContact = Me!cboContacts
   strZahlungspflichtiger1 = Me!txtName1
   strZahlungspflichtiger2 = Nz(Me!txtName2)
   strZahlungspflichtigerIBAN = Me!txtIBANEmpfaenger
   strZahlungspflichtigerBIC = Me!txtBICEmpfaenger
   Set objKontakt = GetContacts.Item(Me!cboContacts)
   Set objAuftraggeber = objKontakt.Accounts(Me!cboAccounts)
   Set objZahlungspflichtiger = AccountErstellen(strZahlungspflichtiger1,
       strZahlungspflichtiger2, strZahlungspflichtigerIBAN, strZahlungspflichtigerBIC)
   Me!txtMeldung = SEPAEinzellastschrift(objKontakt, objZahlungspflichtiger,
       objAuftraggeber, Me!txtBetrag, Me!txtVerwendungszweck,
       Me!txtAusfuehrungsdatum, Me!txtMandatsreferenz, Me!txtMandatsdatum,
       Me!txtGlaeubigerID)
End Sub
```

Die Prozedur prüft zunächst, ob die Funktion *EmpfaengerSpeichern* den Wert *False* zurückliefert. Diese Funktion haben wir bereits in Kapitel *»SEPA-Überweisung« ab Seite 251* vorgestellt – sie prüft, ob bereits eine Bankverbindung mit den angegebenen Daten für den Zahlungspflichtigen angelegt wurde und holt dies gegebenenfalls nach. Falls nicht, fragt die Prozedur, ob der Vorgang gegebenenfalls abgebrochen werden soll. Kapitel 10 Lastschriften mit SEPA

Anschließend ermittelt die Prozedur den Index des Kontakts des Auftraggebers der Lastschrift und speichert diese in der Variablen *intContact* und liest die Daten des Zahlungspflichtigen in entsprechende *String*-Variablen ein.

Über die Funktion *GetContacts* liest die Prozedur dann einen Verweis auf das entsprechende *BACContact*-Objekt ein und speichert diesen in der Variablen *objKontakt*. Das dem im Formular ausgewählten Konto entsprechende *BACAccount*-Objekt wird schließlich mit einer Variablen namens *objAuftraggeber* referenziert.

Fehlt noch ein *BACAccount*-Objekt für den Zahlungspflichtigen. Dieses liefert die Funktion *AccountErstellen*, die das Objekt aus den angegebenen Daten erstellt.

Damit kommen wir dann zum Aufruf der Funktion *SEPAEinzellastschrift*, der die Prozedur alle notwendigen Parameter übergibt. Das Ergebnis dieser Funktion landet im Textfeld *txtMeldung*.

## 10.2.2 Durchführen der Einzellastschrift

Die Funktion SEPAEinzellastschrift erwartet die folgenden Parameter:

- » objContact: BACContact-Objekt des Auftraggebers
- » objZahlungspflichtiger: BACAccount-Objekt für den Zahlungspflichtigen
- » objAuftraggeber: BACAccount-Objekt für den Auftraggeber
- » curBetrag: Einzuziehender Betrag
- » *strVerwendungszweck*: Verwendungszweck
- » datAusfuehrung: Ausführungsdatum
- » strMandatID: Mandatsreferenz
- » datMandat: Datum der Erteilung der Mandatsreferenz
- » strCreditorID: Gläubiger-Identifikationsnummer

### Die Prozedur sieht wie folgt aus und ist im Modul *mdlOBMA\_SEPALastschriften* zu finden:

Public Function SEPAEinzellastschrift(objContact As BACContact,

objZahlungspflichtiger As BACAccount, \_ objAuftraggeber As BACAccount, \_ curBetrag As Currency, \_ strVerwendungszweck As String, \_ datAusfuehrung As Date, \_ strMandatID As String, \_ datMandat As Date, \_ strCreditorID As String)

### Basis-Lastschriften durchführen

Dim objSegmentAnfrage As BACSegment Dim objSegmentVersion As BACSegment Dim objSEPAMessage As BACSepaMessage Dim objSEPAOrder As BACSepaOrder Dim strError As String Dim i As Integer Dim objMessage As BACMessage Dim objBACDialog As BACDialog Dim objTransaction As BACTransaction Dim strAntwort As String Dim strSegment As String strSegment = "HKDSE" Set objSegmentVersion = objAuftraggeber.FindOptimalBPDSegment(strSegment, 0) If Not objSegmentVersion Is Nothing Then Set objHKDSE = objBanking.NewSegment("HKDSE", objHKDSEVersion.Version) Set objSEPAMessage = New BACSepaMessage With objSEPAMessage .Segment = objHKDSE.FromAccount = objZahlungspflichtiger .Descriptor = "pain.008.002.01" .Recipient = objAuftraggeber .ExecutionDate = datAusfuehrung End With Set objSepaOrder = objSepaMessage.Orders.Add With objSepaOrder .ToAccount = objZahlungspflichtiger Amount = curBetrag.CurrencyCode = "EUR" .Purpose = strVerwendungszweck .MandateId = strMandatID .MandateDate = datMandat .CreditorId = strCreditorID End With objSepaMessage.Verify strError If Len(strError) > 0 Then MsgBox strError End If Set objBACDialog = objContact.NewDialogUI objBACDialog.BeginDialog Set objMessage = objBACDialog.ExecuteSegment(objSegmentAnfrage) Set objTransaction = objMessage.Transactions(0)

### Kapitel 10 Lastschriften mit SEPA

Die Prozedur erstellt zunächst die Version des in diesem Fall zu verwendenden Segments *HKDSE*, indem Sie die *FindOptimalBPDSegment*-Funktion für das Auftraggeber-Konto aufruft. Das Ergebnis ist, soweit das Konto das Segment unterstützt, ein *BACSegment*-Objekt mit dem Segment-Typ *HIDSES*. Dessen Inhalt sieht beispielsweise wie folgt aus (Sie erhalten diesen mit der *GetXML*-Funktion):

Die wichtige Information für uns ist der Inhalt des Attributs *Version*, dass wir auch mit der Eigenschaft *Version* abfragen können. Damit erstellen wir nun mit der Funktion *NewSegment* des *BACBanking*-Objekts ein neues Segment auf Basis des Typs *HKDSE* und referenzieren es mit der Objektvariablen *objHKDSE*.

Dieses Objekt ist übrigens annähernd leer (dieses geben Sie im Direktbereich mit dem Befehl *Debug.Print objHKDSE.GetXML* aus):

```
<?xml version="1.0" encoding="UTF-16"?>
<SEGMENT TYPE="HKDSE" VERSION="1"/>
```

### Basis-Lastschriften durchführen

Nun erstellen wir allerdings ein neues Objekt des Typs *BACSepaMessage*. Dieses nimmt alsdann für die Eigenschaft *Segment* das in *objHKDSE* gespeicherte Segment auf. Interessanterweise sorgt dies allerdings dafür, dass das XML-Dokument von oben um weitere Informationen angereichtert wird:

```
<?xml version="1.0" encoding="UTF-16"?>
<SEGMENT TYPE="HKDSE" VERSION="1">
<SEPAPainMessage xmlns:dt="urn:schemas-microsoft-com:datatypes" dt:dt="bin.base64"
Format="object">
</SEPAPainMessage>
</SEGMENT>
```

Interessant deshalb, weil Sie ja eigentlich dem Objekt *objSEPAMessage* eine Information zugewiesen haben. Schauen wir uns doch den aktuellen Inhalt dieses Objekts an und geben diesen Befehl im Direktbereich ein:

? FormatXML(objSEPAMessage.xml)

Dies liefert interessanterweise eine leere Zeichenkette. Nun füllt die Prozedur die Eigenschaft *FromAccount* mit einem Verweis auf das *BACAccount*-Objekt des Zahlungspflichtigen. Auch hier geschieht nichts mit den XML-Inhalten von *objSEPAMessage* oder *objSegmentAnfrage*. Erst beim Zuweisen des Datums für die Eigenschaft *ExecutionDate* füllen sich gleich beide Objekte merklich. Fügen wir noch die Referenz auf den *BACContact* des Auftraggebers hinzu sowie das Ausführungsdatum, sieht beispielsweise der Inhalt von *objSegmentAnfrage* wie folgt aus:

Der Inhalt des Elements *SEPAPainMessage* ist dabei deutlich verkürzt. Er enthält den verschlüsselten Inhalt des Objekts *objSEPAMessage* – der wiederum wie folgt aussieht:

# 11 SEPA-Lastschriften verwalten

Nachdem Sie die Grundlagen zum Thema Lastschriften kennengelernt haben, schauen wir uns nun an, wie Sie einen Anwendungsfall umsetzen können – und zwar die Verwaltung einer Liste von Zahlungspflichtigen samt der dazu benötigten Daten.

Die hier vorgestellten Beispiele sollen Sie später in eigene Anwendungen übernehmen können, daher erstellen wir für die eine oder andere Information eine separate Tabelle – etwa für die Angabe von Daten wie der Gläubiger-ID, der IBAN und der BIC des Zahlungsempfängers.

Davon abgesehen kümmern wir uns aber auch um die Umwandlung eventuell bestehender Einzugsermächtigungen.

# 11.1 Umstellung vom Einzugsermächtigungsverfahren zum SEPA-Basis-Lastschriftverfahren

Wenn Sie bereits vorher mit der herkömmlichen Einzugsermächtigung gearbeitet haben, liegen Ihnen eine Reihe von Daten von Zahlungspflichtigen vor, die auf der Kontonummer und der Bankleitzahl basieren. An dieser Stelle müssen Sie nicht von vorn beginnen, sondern können die vorhandenen Daten und Einzugsermächtigungen nutzen. Dazu sind zunächst zwei Schritte nötig:

- » Konvertierung der Kontonummer und der Bankleitzahl in IBAN und BIC
- » Benachrichtigung des Zahlungspflichtigen über den Ersatz der Lastschrift durch die SEPA-Lastschrift.

# 11.2 Kontonummer und BLZ konvertieren

Für die Konvertierung von Kontonummern und BLZ in IBAN und BIC bietet die DDBAC eine eigene DLL an. Diese heißt *DDBAC.IBAN.Konverter.dll* und wurde mit .NET erstellt. Sie müssen diese von Hand registrieren, und zwar über die Eingabeaufforderung. Diese müssen Sie unbedingt als Administrator ausführen.

Geben Sie im Startmenü unter Suchen | Ausführen den folgenden Befehl ein:

cmd.exe

Das Startmenü zeigt nun die gefundenen Einträge an. Klicken Sie mit der rechten Maustaste auf den entsprechenden Eintrag und wählen Sie aus dem Kontextmenü den Befehl *Als Administrator ausführen …* auf. Hier wechseln Sie nun zu dem Verzeichnis, in dem sich die Datei *Regasm.exe* befindet – auf meinem System beispielsweise hier:

C:\Windows\Microsoft.NET\Framework\v4.0.30319

Nun geben Sie den folgenden Befehl ein, um die entsprechende DLL zu registrieren:

C:\Windows\Microsoft.NET\Framework\v4.0.30319>regasm.exe C:\Daten\Buecher\ OnlinebankingMitAccess\Beispieldateien\DDBAC.IBAN.Konverter.dll /codebase /tlb

Wichtig: Es gibt einen Ordner namens *Framework* und einen namens *Framework64*. Da Sie vermutlich Access in der 32bit-Version verwenden, sollte Sie auch die entsprechende .NET-Version zum Registrieren verwenden, also die im Ordner *Framework* befindliche, nicht die aus *Framework64*.

Regasm.exe meldet dann üblicherweise Folgendes zurück (siehe auch Abbildung 11.1).

Die Typen wurden registriert.

Die Assembly wurde nach "C:\Daten\Buecher\OnlinebankingMitAccess\Beispieldateien\DDBAC. IBAN.Konverter.tlb" exportiert, und die Typbibliothek wurde registriert.

🖾 Administrator: Eingabeaufforderung
C:\Windows\Microsoft.NET\Framework64\v4.0.30319>regasm.exe C:\Daten\Buecher\Onli nebankingMitAccess\Beispieldateien\DDBAC.IBAN.Konverter.dll /codebase /tlb Microsoft .NET Framework Assembly Registration Utility 4.0.30319.18408 für Microsoft .NET Framework, Uersion 4.0.30319.18408 Copyright (C) Microsoft Corporation. Alle Rechte vorbehalten.
Die Typen wurden registriert. Die Assembly wurde nach "C:\Daten\Buecher\OnlinebankingMitAccess\Beispieldateien \DDBAC.IBAN.Konverter.tlb" exportiert, und die Typbibliothek wurde registriert.
C:\Windows\Microsoft.NET\Framework64\u4.0.30319>

Abbildung 11.1: Eingabe des Befehls zum Registrieren der DLL

Wenn Sie nun einmal in das entsprechende Verzeichnis im Windows Explorer schauen, finden Sie dort eine neue Datei namens *DDBAC.IBAN.Konverter.tlb* vor (siehe Abbildung 11.2).

Image: Second system       Image: Second system <td< th=""></td<>								
Organisieren 🔻 🗐 Öffnen 🔻 Freigeben für 🔻 Drucken Brennen Neuer Ordner								
👉 Favoriten	Name	Änderungsdatum	Тур	Größe				
Desktop	DDBAC.IBAN.Konverter.dll	18.03.2014 16:50	Anwendungserwe	3.489 KB				
Downloads	DDBAC.IBAN.Konverter.tlb	13.06.2014 16:50	TLB-Datei	8 KB				
🖳 Zuletzt besucht	ddusers.dat	18.04.2014 19:51	DAT-Datei	2 KB				
iCloud-Fotos	FinTS Segmente.html	12.02.2014 16:28	Chrome HTML Do	882 KB				
AddIns	BCI Segmente.html	27.05.2014 09:27	Chrome HTML Do	767 KB				
lcons	BCI+ Segmente.html	12.02.2014 16:28	Chrome HTML Do	23 KB				
👽 Dropbox	HIRMS.txt	18.04.2014 19:41	Textdokument	1 KB				

Abbildung 11.2: Die .tlb-Datei benötigen Sie zum Einbinden der DLL in das VBA-Projekt.

### Kontonummer und BLZ konvertieren

Wie nutzen Sie diese Konverter nun? Indem Sie die *.tlb*-Datei über die *Durchsuchen*-Schaltfläche des *Verweise*-Dialogs in das VBA-Projekt einbinden (siehe Abbildung 11.3).

Verweise - prjOnlinebanking			
	_		<u> </u>
WlanUi Type Library	٠		Abbrechen
WMPRichPreviewLauncher 1.0 Type Library WMSClientNetManager 1.0 Type Library WorkenacePuntime 1.0 Type Library			Durchsuchen
WPC private 1.0 Type Library WPDsp 1.0 Type Library		+	
WSHControllerLibrary		Priorität	Hilfe
WUAPI 2.0 Type Library		+	
XPS_SHL_DLL 1.0 Type Library			
	Ŧ		
۰			
DataDesign DDBAC IBAN Konverter			
Pfad: C:\DATEN\BUECHER\ONLINEBANKIN	NGN	ITACCESS	BEISPIELDATEIE
Sprache: Voreinstellung			

Abbildung 11.3: Hinzufügen des IBAN-Konverters

Wenn Sie nun im VBA-Editor mit *F2* den Objektkatalog einblenden und im oberen Kombinationsfeld den Eintrag *DDBAC\_IBAN\_Konverter* auswählen, finden Sie die Objekte und Methoden der DLL vor (siehe Abbildung 11.4).

📲 Objektkatalog				- • ×	
DDBAC_IBAN_Konverter	< ₩ ^	<u> B</u> > ?			
Suchergebnisse					
Bibliothek	Klas	se	Element		
Klassen • <global></global>		Elemente von 'BAClbank	(onverter'		
BACIbanKonverter		RetrieveAccountNumberAndBankCode			
BACIbanKonverterResponse		RetrieveBicForBankCode			
BACIDANKONVERTERStatus		RetrieveBicForiban			
		SetBlzScript			
Class BACIbanKonverter Element von <u>DDBAC IBAN Konv</u>	<u>erter</u>				

Abbildung 11.4: Die Objekte der DLL mit ihren Methoden

Schauen wir uns zunächst eine Übersicht der verfügbaren Methoden an:

- » Function CheckIban(ByVal sIban As String) As BACIbanKonverterStatus: Prüft die angegebene IBAN.
- » Function RetrieveAccountNumberAndBankCode(ByVal sIban As String) As BACIbanKonverterResponse: Ermittelt die Kontonummer und die Bankleitzahl für die angegebene IBAN.
- » Function RetrieveBicForBankCode(ByValsBankCode AsString) AsBACIbanKonverterResponse: Ermittelt die BIC für die angegebene Bankleitzahl.
- » Function RetrieveBicForlban(ByVal slban As String) As BAClbanKonverterResponse: Ermittelt die BIC für die angegebene IBAN.
- » Function RetrievelbanAndBic(ByVal sBankCode As String, ByVal sAccountNumber As String, ByVal sCheckSumMethod As String) As BAClbanKonverterResponse: Ermittelt IBAN und BIC für die angegebene BLZ und Kontonummer.

Das zweite Objekt, das die DLL offeriert, heißt *BACIbanKonverterResponse*. Es wird von einigen der oben genannten Funktionen zurückgeliefert und stellt alle notwendigen Informationen bereit. Dazu dienen die folgenden Eigenschaften:

- » AccountNumber: String-Eigenschaft mit der Kontonummer
- » BankCode: Bankleitzahl (String)
- » Bic: BIC (String)
- » Iban: IBAN (String)
- » *Status*: Status (*Integer*-Konstanten aus *BACIbanKonverterStatus*). Eine Liste der möglichen Werte finden Sie weiter unten.
- » StatusMessage: Statusmeldung (String)
- » Success: Boolean-Eigenschaft, die angibt, ob die Methode erfolgreich war.

Die folgenden Statuswerte werden entweder direkt von einer Funktion zurückgeliefert (beispielsweise von *Checklban*) oder sind als Eigenschaft *Status* des als Antwort gesendeten Objekts *BAClbanResponse* verfügbar:

- » BACIbanKonverterStatus\_OK (0)
- » BACIbanKonverterStatus\_ACCOUNT\_NUMBER\_REPLACED (1)
- » BACIbanKonverterStatus\_INVALID\_BANK\_CODE (10)
- » BACIbanKonverterStatus\_INVALID\_ACCOUNT\_NUMBER (11)
- » BACIbanKonverterStatus\_EXCLUDED\_ACCOUNT\_NUMBER (12)
- » BACIbanKonverterStatus\_BANK\_CODE\_REPLACED (13)

### Kontonummer und BLZ konvertieren

- » BACIbanKonverterStatus\_BANK\_CODE\_MARKED\_FOR\_DELETION (14)
- » BACIbanKonverterStatus\_BANK\_CODE\_FOLLOW\_UP\_EXISTS (15)
- » BACIbanKonverterStatus\_MAL\_FORMED\_IBAN (20)
- » BACIbanKonverterStatus\_INVALID\_IBAN (21)
- » BAClbanKonverterStatus\_INVALID\_BIC (22)
- » BACIbanKonverterStatus\_NO\_BIC\_FOUND (23)
- » BACIbanKonverterStatus\_MISSING\_ACCOUNT\_NUMBER\_FOR\_BIC (24)
- » BACIbanKonverterStatus\_CHECKSUM\_NOT\_CHECKED (30)
- » BACIbanKonverterStatus\_INVALID\_ACCOUNT\_FOR\_BANK (40)
- » BACIbanKonverterStatus\_IBAN\_CALCULATION\_FAILED (50)
- » BACIbanKonverterStatus\_IBAN\_WITH\_BEST\_GUESS\_CALCULATION (51)
- » BACIbanKonverterStatus\_IBAN\_WITH\_STANDARD\_CALCULATION (52)
- » BACIbanKonverterStatus\_TECHNICAL\_ERROR (99)

## 11.2.1 Beispiele für den Einsatz der DLL

Im Folgenden finden Sie einige nützliche Beispiele für den Einsatz der Methoden der DLL.

### **IBAN** aus **BLZ** und Kontonummer ermitteln

Der gängigste Fall dürfte die Ermittlung von IBAN und/oder BIC aus einer gegebenen Kombination aus Bankleitzahl und Kontonummer sein. Dies ist für das nachfolgende Beispiele zur Umstellung von der herkömmlichen Lastschrift auf die SEPA-Lastschrift wichtig.

Im folgenden Beispiel verwenden wir eine Funktion namens *IBANAusBLZUndKontonummer*, um die IBAN aus den per Parameter übergebenen Werten zu ermitteln (siehe Modul *mdlIBANRechner*):

```
If .Success = True Then
IBANAusBLZUndKontonummer = .IBAN
End If
End With
End With
End Function
```

Dabei erstellt die Prozedur zunächst ein neues Objekt auf Basis der Klasse BAClbanKonverterResponse der Bibliothek DDBAC\_IBAN\_Konverter. Dieses liefert die Methode RetrievelBANAndBic, welche die beiden in den Variablen strBLZ und strKontonummer enthaltenen Informationen sowie eine leere Variable namens strChecksum erwartet. Das Ergebnis dieser Funktion landet in der Variablen objICResponse. Hier prüfen wir zunächst den Wert der Eigenschaft Success. Ist dieser True, wurde ein Wert ermittelt und wir können diesen über die Eigenschaft IBAN einlesen und als Funktionswert zurückgeben. Beispielaufruf:

```
Debug.Print IBANAusBLZUndKontonummer("35040038", "411890700")
DE38350400380411890700
```

### BIC aus der Bankleitzahl ermitteln

Eine zweite Funktion heißt *BICAusBLZ* und ermittelt aus der Bankleitzahl die BIC. Diese Funktion ist ähnlich aufgebaut wie die erste, allerdings erwartet sie lediglich die BLZ als Parameter:

```
Public Function BICAusBLZ(strBLZ As String) As String
  Dim objIC As DDBAC_IBAN_Konverter.BACIbanKonverter
  Dim objICResponse As DDBAC_IBAN_Konverter.BACIbanKonverterResponse
  Set objIC = New DDBAC_IBAN_Konverter.BACIbanKonverter
  With objIC
    Set objICResponse = .RetrieveBicForBankCode(strBLZ)
    With objICResponse
        If .Success = True Then
            BICAusBLZ = .BIC
        End If
        End With
    End With
  End With
  End Function
```

Nach dem Erstellen des BACIbanKonverter-Objekts verwenden wir diesmal die Methode RetrieveBicForBankCode. Dies liefert wiederum ein Objekt des Typs BACIbanKonverterResponse zurück, welches mit der Eigenschaft BIC den gesuchten Wert offeriert. Leseprobe! Vollständiges Buch erhältlich unter www.amvshop.de im André Minhorst Verlag.

### Kontonummer und BLZ konvertieren

### **IBAN prüfen**

Wenn Sie eine Reihe von Kontonummern und Bankleitzahlen erhalten haben, zu denen bereits eine IBAN mitgeliefert wurde, können Sie prüfen, ob die IBAN korrekt ist. Dazu verwenden Sie die folgende Funktion namens *IBANPruefen*. Sie erwartet die IBAN als Parameter:

```
Public Function IBANPruefen(strIBAN As String) As Boolean
  Dim objIC As DDBAC_IBAN_Konverter.BACIbanKonverter
  Dim intStatus As DDBAC_IBAN_Konverter.BACIbanKonverterStatus
  Set objIC = New DDBAC_IBAN_Konverter.BACIbanKonverter
  With objIC
    intStatus = .CheckIban(strIBAN)
    Select Case intStatus
      Case BACIbanKonverterStatus_OK
        IBANPruefen = True
        Case Else
        IBANPruefen = False
        End Select
        End With
End Function
```

Diesmal verwenden wir die Funktion *CheckIBAN* des Objekts *objIC*. Diese liefert nun kein Objekt des Typs *BACIbanKonverterResponse* zurück, sondern gleich das Ergebnis in Form eines Wertes der Enumeration *BACIbanKonverterStatus*. In diesem Fall sollte die Konstante *BACIbanKonverterStatus\_OK* zurückgeliefert werden, damit da Ergebnis korrekt ist. Beispielaufruf:

```
Debug.Print IBANPruefen("DE24350400380415109800")
Wahr
```

Ob eine IBAN zu Kontonummer und Bankleitzahl passt, können Sie mit der weiter oben vorgestellten Funktion *IBANAusBLZUndKontonummer* ermitteln. Dort vergleichen Sie dann einfach das Funktionsergebnis mit der entsprechenden IBAN und erhalten den Wert *Wahr* oder *Falsch* als Ergebnis:

```
Debug.Print IBANAusBLZUndKontonummer("35040038", "4151098") = "DE24350400380415109800"
Wahr
```

### **BIC aus der IBAN ermitteln**

Liegt die IBAN vor und Sie benötigen noch die BIC, verwenden Sie die Funktion *BICAusIBAN*. Diese erwartet die IBAN als *String*-Parameter:

```
Public Function BICAusIBAN(strIBAN As String) As String
Dim objIC As DDBAC_IBAN_Konverter.BACIbanKonverter
Dim objICResponse As DDBAC_IBAN_Konverter.BACIbanKonverterResponse
```

```
Set objIC = New DDBAC_IBAN_Konverter.BACIbanKonverter
Set objICResponse = objIC.RetrieveBicForIban(strIBAN)
With objICResponse
Select Case .status
Case BACIbanKonverterStatus_OK
BICAusIBAN = .BIC
Case Else
BICAusIBAN = ""
End Select
End With
End Function
```

Die hier verwendete Funktion *RetrieveBicForlban* liefert wiederum ein Objekt des Typs *BAClban-KonverterResponse* zurück, dem Sie dann über die Eigenschaft *BIC* die gewünschte Information entnehmen können.

### Kontonummer aus der IBAN ermitteln

Gegebenenfalls liegt Ihnen die IBAN vor, Sie möchten aber die Kontonummer ermitteln. Auch dies lässt sich mit der Klasse *BAClbanKonverter* abbilden. Die Funktion *KontonummerAusIBAN* erwartet die IBAN als Parameter:

```
Public Function KontonummerAusIBAN(strIBAN As String) As String
   Dim objIC As DDBAC IBAN Konverter.BACIbanKonverter
   Dim objICResponse As DDBAC IBAN Konverter, BACIbanKonverterResponse
   Dim intStatus As DDBAC IBAN Konverter.BACIbanKonverterStatus
   Set objIC = New DDBAC IBAN Konverter.BACIbanKonverter
   Set objICResponse = objIC.RetrieveAccountNumberAndBankCode(strIBAN)
   With objICResponse
       Select Case .status
           Case BACIbanKonverterStatus OK
               KontonummerAusIBAN = .AccountNumber
           Case Else
               MsgBox intStatus
               KontonummerAusIBAN = ""
       Fnd Select
   End With
End Function
```

Die Funktion ruft die Funktion *RetrieveAccountNumberAndBankCode* des Objekts *objIC* auf und erhält ein Objekt des Typs *BAClbanKonverterResponse* als Rückgabewert zurück. Diesem entnimmt die Funktion mit der Eigenschaft *AccountNumber* die Kontonummer.

Kontonummer und BLZ konvertieren

### 11.2.2 Testen der Funktionen

Diese Funktionen können Sie komfortabel mit dem Formular *frmIBANRechner* ausprobieren. Auf der linken Seite dieses Formulars finden Sie vier Textfelder zur Eingabe von Bankleitzahl, Kontonummer, IBAN und BIC. Mit der Eingabe eines jeden Zeichens werden die Felder rechts aktualisiert und erhalten die ermittelten Informationen. Wenn Sie etwa links die Kontonummer und die Bankleitzahl eingeben, erscheinen auf der rechten Seite die IBAN und die BIC (siehe Abbildung 11.5).

= frmIBANRechner	I frmIBANRechner					
IBAN-Rechner						
Eingabe:	Erg	ebnis:				
BLZ: 35040038	BLZ	:				
Kontonummer: 411890700	Kor	ntonummer:				
IBAN:	IBA	N:	DE38 3504 00	03 <mark>8 0</mark> 4:	11 890	7 00
BIC:	BIC		COBADEFFXX	ĸ		
К						

Abbildung 11.5: Formular zum Testen der Umwandlungsfunktionen

Wenn der Benutzer beispielsweise die Zeichenkette im Textfeld *txtBLZ* ändert, löst er damit die folgende Prozedur aus:

```
Private Sub txtBLZ_Change()
   strBIC = Replace(Nz(Me!txtBIC), " ", "")
   strBLZ = Replace(Me!txtBLZ.Text, " ", "")
   strIBAN = Replace(Nz(Me!txtIBAN), " ", "")
   strKontonummer = Replace(Nz(Me!txtKontonummer), " ", "")
   Berechnen
End Sub
```

Diese speichert den Text aus dem aktuellen Textfeld sowie der anderen Textfelder auf der linken Seite des Formulars in den folgenden Variablen:

```
Dim strIBAN As String
Dim strBIC As String
Dim strBLZ As String
Dim strKontonummer As String
```

Danach ruft sie die Prozedur Berechnen auf, die wie folgt aussieht:

```
Private Sub Berechnen()
```

```
Me!txtBICResult = BICAusBLZ(strBLZ)
If IsNull(Me!txtBICResult) Then
    Me!txtBICResult = BICAusIBAN(strIBAN)
End If
Me!txtIBANResult = IBANAusBLZUndKontonummer(strBLZ, strKontonummer)
Me!txtBLZResult = BLZAusIBAN(strIBAN)
Me!txtKontonummerResult = KontonummerAusIBAN(strIBAN)
End Sub
```

Diese Prozedur benutzt die weiter oben vorgestellten Funktionen, um die Textfelder auf der rechten Seite des Formulars mit den aktuell ermittelbaren Informationen zu füllen.

Die übrigen drei Textfelder auf der linken Seite des Formulars rufen ähnliche Prozeduren auf, die jeweils die Inhalte aller Textfelder speichern und die Prozedur *Berechnen* aufrufen.

# 11.3 Massenweise Konvertierung zu IBAN/BIC

Wenn Sie sich selbst eine Access-Anwendung zur Verwaltung eines Sport-, Kleingarten oder sonstigen Vereins gebaut haben und damit bereits die Lastschriften eingezogen haben, können Sie dies auch unter SEPA mit relativ wenig Arbeit bewerkstelligen. Als Erstes wollen wir dazu die notwendigen Kontodaten konvertieren.

Um dies flexibel zu gestalten, erstellen wir ein Formular, mit dem Sie die Tabelle mit den relevanten Daten auswählen, die Felder mit BLZ und Kontonummer markieren und angeben, in welche Felder IBAN und/oder BIC geschrieben werden sollen.

Das Formular heißt *frmIBANKonverter* und sieht in der Entwurfsansicht wie in Abbildung 11.6 aus. Es enthält ein Kombinationsfeld namens *cboTabelle*, welches alle Tabellen der aktuellen Datenbank anzeigt. Die übrigen Kombinationsfelder *cboBLZ*, *cboKontonummer*, *cboIBAN* und *cboBIC* dienen der Auswahl des Feldes der mit *cboTabelle* selektierten Tabelle, das die jeweilige Information anzeigt beziehungsweise aufnehmen soll.

Außerdem enthält das Formular ein Unterformular namens *sfmIBANKonverter*, das alle Datensätze der ausgewählten Tabelle liefert und für die enthaltenen Kontonummer und Bankleitzahlen die berechneten IBAN und BIC anzeigt. Die Schaltfläche *cmdKonvertieren* schließlich trägt alle ermittelten IBAN und BIC in die dafür festgelegten Felder ein.

Massenweise Konvertierung zu IBAN/BIC

international in		23					
1	· 15 · i · 1	i6 · 🔺					
✓ Formularkopf							
IBAN-Konvertierer							
Detailbereich		_					
Tabelle: Ungebunden							
1 BLZ-Feld: Ungebunden							
Kontonummer-Feld: Ungebunden		=					
IBAN-Feld: Ungebunden							
3 BIC-Feld:							
4 	14 · •						
Detailbereich	≡						
BLZ: BLZ							
Kontonummer: Kontonummer							
IBAN: IBAN							
7 2 BIC: BIC	_						
- Status: Status							
Konvertieren							
✓ Formularfuß							
- IX ок							
		-					

Abbildung 11.6: IBAN-Konverterformular in der Entwurfsansicht

Für das Kombinationsfeld *cboTabelle* hinterlegen wir eine Datenherkunft, die alle Tabellen der aktuellen Datenbank liefert. Diese verwendet die Systemtabelle *MSysObjects* als Quelle. Deren Feld *Name* liefert die Namen der enthaltenen Objekte. Die Tabellen filtern wir heraus, indem wir für das Feld *Type* das Kriterium *IN* (1;4;6) festlegen. 1 steht für Tabellen der gleichen Datenbank, 4 für per ODBC verknüpfte Tabellen und 6 für per DAO verknüpfte Tabellen (siehe Abbildung 11.7).

Außerdem stellen wir als Kriterium für das Feld *Name* noch den folgenden Ausdruck ein, damit das Kombinationsfeld keine Systemtabellen anzeigt:

Nicht Wie 'MSYS\*'

frmIBANC		• **			
MSysObject * Conni Datab DateC DateL Flags Foreig					
					а.
Feld:	Name	Type		<b>A</b>	
Tabelle:	MSvsObjects	MSvsObjects			
Sortierung:	Aufsteigend				
Anzeigen:					
Kriterien:	Nicht Wie 'MSYS*'	In (1;4;6)			
oder:				•	

Abbildung 11.7: Datensatzherkunft zur Anzeige aller Tabellen einer Datenbank

Wenn der Benutzer eine der Tabellen der Datenbank auswählt, sollen die Felder dieser Tabelle als Datensatzherkunft der übrigen Kombinationsfelder *cboBLZ*, *cboKontonummer*, *cboIBAN* und *cboBIC* angezeigt werden. Aber wir wollen noch mehr: Typischerweise sind die Feldnamen ja recht aussagekräftig. Das Feld, welches die Bankleitzahl enthält, heißt meist auch *BLZ*. Gleiches gilt auch für *BIC* und *IBAN*. Bei der Kontonummer gibt es auch nicht allzuviele Varianten. Also wollen wir dafür sorgen, dass die Felder mit dem jeweiligen Namen auch gleich in die entsprechenden Kombinationsfelder eingetragen werden. Zunächst jedoch benötigen wir überhaupt eine passende Tabelle. Diese heißt in unserer Beispieldatenbank *tblMitglieder* und sieht im Entwurf wie in Abbildung 11.8 aus.

Die Tabelle enthält neben den Feldern für die Bankleitzahl, die Kontonummer, die IBAN und die BIC auch gleich noch Felder für die Mandatsreferenznummer und das Mandatsdatum.

### Massenweise Konvertierung zu IBAN/BIC

Feldname	Felddatentyp	Beschreibung		
MitgliedID	AutoWert	Primärschlüsselfeld der Tabelle		
AnredeID	Zahl	Fremdschlüsselfeld zur Tabelle tblAnreden		
Vorname	Text	Vorname des Mitglieds		
Nachname	Text	Nachname des Mitglieds		
Strasse	Text	Strasse des Mitglieds		
PLZ	Text	PLZ des Mitglieds		
Ort	Text	Ort des Mitglieds		
EMail	Text	E-Mail-Adresse		
Eintrittsdatum	Datum/Uhrzeit	Tag des Eintritts in den Verein		
Austrittsdatum	Datum/Uhrzeit	Tag des Austritts aus dem Verein		
BeitragsgruppeID	Zahl	Fremdschlüsselfeld zur Tabelle tblBeitragsgruppen		
Kontonummer	Text	Kontonummer		
BLZ	Text	Bankleitzahl		
Kontoinhaber	Text	Name des Kontoinhabers		
BIC	Text	BIC		
IBAN	Text	IBAN		
Mandatsreferenznummer	Text	Mandatsreferenznummer für SEPA-Lastschriften		
Mandatsdatum	Datum/Uhrzeit	Datum, an dem das Mandat unterschrieben wurde		

Abbildung 11.8: Beispieltabelle einer Mitgliederverwaltung

Nun wollen wir eine Funktionalität bereitstellen, die nach dem Auswählen einer Tabelle mit dem Kombinationsfeld *cboTabelle* ausgelöst wird. Dazu stellen Sie die Ereigniseigenschaft *Nach Aktualisierung* dieses Steuerelements auf *[Ereignisprozedur]* ein und klicken auf die Schaltfläche rechts neben der Eigenschaft. Die nun erscheinende leere Ereignisprozedur füllen Sie wie folgt:

```
Private Sub cboTabelle AfterUpdate()
    Dim db As DAO.Database
    Dim tdf As DAO.TableDef
    Dim fld As DAO.Field
    Dim strTabelle As String
    Set db = CurrentDb
    strTabelle = Me!cboTabelle
    With Me!cboBLZ
        .RowSourceType = "Field List"
        .RowSource = strTabelle
    Fnd With
    With Me!cboKontonummer
        .RowSourceType = "Field List"
        .RowSource = strTabelle
    End With
    With Me!cboIBAN
        .RowSourceType = "Field List"
        .RowSource = strTabelle
    Fnd With
```

```
With Me!cboBIC
        .RowSourceType = "Field List"
        .RowSource = strTabelle
    End With
    Set tdf = db.TableDefs(strTabelle)
    For Fach fld In tdf.Fields
        If InStr(1, fld.Name, "Kontonummer") Then
            Me!cboKontonummer = fld.Name
       End If
        If InStr(1, fld.Name, "BLZ") Then
           Me!cboBLZ = fld.Name
        Fnd If
        If InStr(1, fld.Name, "IBAN") Then
            Me!cboIBAN = fld.Name
        Fnd If
        If InStr(1, fld.Name, "BIC") Then
            Me!cboBIC = fld.Name
        Fnd If
   Next fld
   VorschauAnzeigen
End Sub
```

Die Prozedur trägt zunächst den Wert des Kombinationsfeldes *cboTabelle* in die Variable *strTabelle* ein. Dann legt die Prozedur für die Eigenschaft *RowSourceType* der vier übrigen Kombinationsfelder den Wert *Field List* fest, was dem Wert *Feldliste* für die Eigenschaft *Herkunftsart* entspricht. Die Eigenschaft *RowSource* füllt die Prozedur mit dem Namen der soeben ausgewählten Tabelle. Dadurch zeigen die Kombinationsfelder nun die Felder der gewählten Tabelle an (siehe Abbildung 11.9).

### Massenweise Konvertierung zu IBAN/BIC

😑 frmIBANConverte	r	}
	Konvertierer	
Tabelle:	tblMitglieder	•
BLZ-Feld:	BLZ	
Kontonummer-Feld:	Vorname Nachname	^
IBAN-Feld:	Strasse	
BIC-Feld:	PLZ Ort	
BLZ - 35040038 76010085 12121212	EMail Eintrittsdatum Austrittsdatum BeitragsgruppeID Kontonummer	BIC OBADEF BNKDEF
*	BLZ Kontoinhaber BIC IBAN	
Datensatz: H 🚽 1 ve	Mandatsdatum	

Abbildung 11.9: Auswahl der Felder für Kontonummer, BLZ, IBAN und BIC

Das ist aber nur die halbe Miete – etwas Komfort fehlt noch. Also füllt die Prozedur eine Variable namens *tdf* mit einem Verweis auf das *TableDef*-Objekt zur ausgewählten Tabelle. Wozu das? Um gleich alle Felder der Tabelle in einer *For Each*-Schleife zu durchlaufen und dabei zu prüfen, ob das jeweils aktuelle Feld einen der Ausdrücke *Kontonummer, BLZ, IBAN* oder *BIC* enthält. Ist dies der Fall, stellt die Prozedur das entsprechende Kombinationsfeld gleich auf den entsprechenden Feldnamen ein. Das Ergebnis sieht etwa wie in Abbildung 11.10 aus – und dazu ist allein die Auswahl der Tabelle *tblMitglieder* notwendig.

= frmIBANConverte		23		
iban-i				
Tabelle:	tblMitglieder	•		
BLZ-Feld:	BLZ	•		
Kontonummer-Feld:	Kontonummer	•		
IBAN-Feld:	IBAN	•		
BIC-Feld:	BIC	•		

Abbildung 11.10: Anzeige aller notwendigen Informationen nach der Auswahl der Zieltabelle

Schließlich ruft die Prozedur die Routine VorschauAnzeigen auf:

```
Private Sub VorschauAnzeigen()
Dim strSQL As String
```

```
Dim strTabelle As String
strTabelle = Me!cboTabelle
If Not Len(Nz(Me!cboBIC)) * Len(Nz(Me!cboIBAN)) * _
Len(Nz(Me!cboBLZ)) * Len(Nz(Me!cboKontonummer)) = 0 Then
strSQL = "SELECT " & Me!cboKontonummer & " AS Kontonummer, " & Me!cboBLZ _
& " AS BLZ, IBANAusBLZUndKontonummer(Nz([" & Me!cboBLZ & "],''),Nz([" _
& Me!cboKontonummer & "],'')) AS " & Me!cboIBAN _
& ", BICAusBLZ(Nz([" & Me!cboBLZ & "],'')) AS " & Me!cboBIC _
& ", IIf(Len([" & Me!cboIBAN & "])*Len([" _
& Me!cboBIC & "])>0, ""OK"", ""Fehler"") AS Status FROM " & strTabelle & ";"
Me!sfmKonvertierung.Form.RecordSource = strSQL
End If
End Sub
```

Diese prüft zunächst, ob der Benutzer für die vier Kombinationsfelder *cboBLZ*, *cboKontonummer*, *cboIBAN* und *cboBIC* je ein Feld ausgewählt hat. Dazu multipliziert sie einfach die Längen der enthaltenen Zeichenketten. Ist mindestens eines der Kombinationsfelder leer, liefert die *If... Then*-Bedingung den Wert *False* und die darin enthaltenen Befehle werden nicht ausgeführt. Anderenfalls stellt die Prozedur eine SQL-Abfrage zusammen, welche auf der gewählten Tabelle und den vier angegebenen Feldern basiert. Die beiden Feldnamen für die Felder mit der Kontonummer und der BLZ werden einfach aus den beiden Kombinationsfeldern *cboKontonummer* und *cboBLZ* bezogen.

Das Feld zur Darstellung des Wertes für die IBAN wird als berechnetes Feld ausgeführt. Der Ausdruck für dieses Feld entspricht einem Aufruf der weiter oben bereits vorgestellten VBA-Funktion *IBANAusBLZUndKontonummer* und verwendet die Inhalte der beiden Felder für die Kontonummer und die BLZ als Parameter. Die BIC wird mit der VBA-Funktion *BICAusBLZ* und dem in *cboBLZ* angegebenen Feld ermittelt.

Fehlt noch ein letztes Feld, das angibt, ob die beiden Werte für IBAN und BIC korrekt ermittelt werden konnten. Dieses prüft, ob die beiden Felder für die IBAN und die BIC gefüllt sind und gibt entsprechend einen der Werte *OK* oder *Fehler* aus. Der vollständige SQL-Ausdruck für unsere Beispieltabelle sieht schließlich wie folgt aus:

```
SELECT Kontonummer, BLZ,
IBANAusBLZUndKontonummer(Nz([BLZ],'').Nz([Kontonummer],'')) AS IBAN,
BICAusBLZ(Nz([BLZ],'')) AS BIC,
IIf(Len([IBAN])*Len([BIC])>0."OK"."Fehler") AS Status FROM tblMitglieder;
```

Dieser SQL-Ausdruck landet schließlich als Datenherkunft im Unterformular *sfmIBANKonverter*. Das Unterformular sieht im Entwurf wie in Abbildung 11.11 aus. Da dieses die fixen Steuerelementinhalte *BLZ*, *Kontonummer*, *IBAN* und *BIC* enthält, werden die Felder der Datenherkunft dieses Unterformulars mit der entsprechenden Bezeichnung per *AS-Klausel v*ersehen (zum Beispiel *BICAusBLZ*(*Nz*([*BLZ*], '')) *AS BIC*).

### Massenweise Konvertierung zu IBAN/BIC

==	sfmKonvertierung					23
	+ + + 1 + + + 2 + + + 3	4 5 6 .	1 * 7 * 1 * 8 * 1 * 9 * 1 * 10 * 1 * 11	· · · 12 ·	1 13 1	1 •
	Detailbereich					
<u> </u>	BLZ:	BLZ				_
1	Kontonummer:	Kontonummer				
	IBAN:	IBAN				
	BIC:	BIC				
3	Status:	Status				
1						
4		101				

Abbildung 11.11: Das Unterformular in der Entwurfsansicht

Nun kann es natürlich geschehen, dass die Felder mit den benötigten Informationen andere Namen als geplant haben. In diesem Fall muss der Benutzer doch noch ein oder mehrere Kombinationsfelder von Hand füllen. Dies soll dann wiederum jeweils die Prozedur *VorschauAnzeigen* auslösen. Dazu fügen wir dem Formular noch die folgenden vier Ereigniseigenschaften hinzu:

```
Private Sub cboBIC_AfterUpdate()
    VorschauAnzeigen
End Sub
Private Sub cboBLZ_AfterUpdate()
    VorschauAnzeigen
End Sub
Private Sub cboIBAN_AfterUpdate()
    VorschauAnzeigen
End Sub
Private Sub cboKontonummer_AfterUpdate()
    VorschauAnzeigen
End Sub
```

Fehlt noch die Schaltfläche, welche die in der Vorschau angezeigten IBAN- und BIC-Werte tatsächlich in die Zielfelder schreift. Diese Prozedur löst der Benutzer durch einen Klick auf die Schaltfläche *cmdKonvertieren* aus, was die folgende Prozedur startet:

```
Private Sub cmdKonvertieren_Click()
Dim db As DAO.Database
Dim rst As DAO.Recordset
Dim strIBAN As String
Dim strBIC As String
```

```
Dim strKontonummer As String
   Dim strBLZ As String
   Dim lngAktualisiert As Long
   Dim strTabelle As String
   strTabelle = Me!cboTabelle
   Set db = CurrentDb
   Set rst = db.OpenRecordset("SELECT " & Me!cboBIC & ", " & Me!cboBLZ & ", "
       & Me!cboIBAN & ", " & Me!cboKontonummer & " FROM " & strTabelle, dbOpenDynaset)
   Do While Not rst.EOF
       strBLZ = Nz(rst(Me!cboBLZ), "")
       strKontonummer = Nz(rst(Me!cboKontonummer). "")
        If Not Len(strBLZ) * Len(strKontonummer) = 0 Then
           strIBAN = IBANAusBLZUndKontonummer(strBLZ, strKontonummer)
           strBIC = BICAusBLZ(strBLZ)
           If len(strIBAN) > 0 Then
               rst.Edit
               rst(Me!cboIBAN) = strIBAN
               rst(Me!cboBIC) = strBIC
               lngAktualisiert = lngAktualisiert + 1
               rst.Update
           Fnd If
       Fnd If
       rst.MoveNext
   Loop
   MsgBox "Es wurden " & lngAktualisiert & "/" & rst.RecordCount
       & " Datensätzen aktualisiert."
Fnd Sub
```

Die Prozedur speichert den Namen der zu verwendenden Tabelle in der Variablen *strTabelle* und öffnet dann ein Recordset auf Basis dieser Tabelle. Das Recordset enthält die vier in den übrigen Kombinationsfeldern angegebenen Felder. Die Prozedur durchläuft dann alle Datensätze dieser Tabelle und prüft zunächst, ob sowohl das für die Kontonummer als auch das für die Bankleitzahl angegebene Feld gefüllt ist (die Werte landen dazu in den beiden Variablen *strKontonummer* und *strBLZ*). Dann ermittelt die Prozedur mit den bereits bekannten Funktionen die IBAN und die BIC. Nur wenn die IBAN ermittelt werden konnte, trägt die Prozedur die beiden Werte aus *strIBAN* und *strBIC* in die entsprechenden Felder der Zieltabelle ein. Schließlich erhöht sie die Zählervariable in diesem Fall um eins, sodass nach dem Durchlaufen aller Datensätze die Anzahl der aktualisierten Datensätze per Meldungsfenster ausgegeben werden kann.

**Umstellung auf SEPA-Lastschriften** 

# 11.4 Umstellung auf SEPA-Lastschriften

Wenn Sie bereits herkömmliche Lastschriften eingezogen haben, können Sie diese ohne Probleme in die neue SEPA-Lastschrift überführen. Die Zahlungspflichtigen informieren Sie vor der Umstellung auf die SEPA-Lastschrift dann mit einem entsprechenden Anschreiben. Ein solches haben wir als Bericht angelegt, der auf die Daten des obigen Datenmodells zugreift. Dieses haben wir noch um zwei Tabellen erweitert, welche die Anreden und die Beitragsgruppen bereitstellen. Insgesamt sieht das Datenmodell nun wie in Abbildung 11.12 aus.



Abbildung 11.12: Datenmodell für die Mitgliedsverwaltung

## 11.4.1 Stammdaten des Empfängers speichern

Neben diesen Tabellen, welche die Daten der Zahlungspflichtigen enthalten, soll die Datenbank auch die Daten des Zahlungsempfängers verwalten. Dazu erstellen wir eine Tabelle namens *tblOptionenLastschrift*, welche die Basisdaten des Auftraggebers enthält. Diese sieht im Entwurf wie folgt aus:

Í		tblOptionenLastschrift	:		Σ	3
	2	Feldname	Felddatentyp	Beschreibung		
		GlaeubigerID	Text	Gläubiger-Identifikationsnummer		
		IBAN	Text	IBAN des Auftraggebers		
		BIC	Text	BIC des Auftraggebers		
L					 	

Abbildung 11.13: Stammdaten des Zahlungsempfängers

# 12 Fehlerbehandlung

Wir wollen nun noch zeigen, wo durch den Zugriff auf die Server der Bankinstitute Fehler auftauchen können und wie Sie diese behandeln. Dies erledigen wir zunächst anhand des einfachen Abrufs des Kontostands.

Mit der Veröffentlichung dieses Buchs stehen Ihnen als Leser zwei Testkonten zur Verfügung, mit denen Sie die verschiedenen Geschäftsvorfälle ausprobieren können. Die Konten bieten nicht alle Geschäftsvorfälle an, aber dennoch finden Sie hier ausreichend Testmöglichkeiten. Die Zugangsdaten finden Sie weiter vorn unter »Bankverbindung zum Testen der Anwendung« ab Seite 39.

# 12.1 Test-PIN und -TAN

Zur Vereinfachung der Tests bieten die Testkonten die Möglichkeit, durch die Eingabe von bestimmten PIN- und TAN-Nummern einige Fehler auszulösen. Nachfolgend finden Sie eine Auflistung dieser Nummern – zunächst einige PIN-Nummern:

- » 01111: PIN gesperrt
- » 02222: PIN muss vor erstmaliger Anmeldung geändert werden.
- » 03333: Abbruch des Dialogs und Rückmeldung mit Code 9333.
- » 99998: Vorläufige PIN-Sperre.
- » 09999: Verarbeitungssystem nicht verfügbar.
- » Oxxxx: PIN ungültig.
- » 99999: PIN gültig.
- » *yxxxx*: PIN-Prüfung erfolgreich (y<>0).

PIN-Tests zur Demobank bei PIN-Änderung:

- » Oxxxx: PIN ungültig.
- » yxxxx: PIN-Prüfung erfolgreich (y<>0).

TAN-Tests zur Demobank:

- » 099999: TAN verwendet.
- » 088888: TAN-Liste gesperrt.
- » 009951: TAN ungültig (bei zwei-Schritt-TAN-Verfahren).

Kapitel 12 Fehlerbehandlung

- » 009210: Auftrag nicht gefunden (bei zwei-Schritt-TAN-Verfahren).
- » Oxxxxx: TAN ungültig.
- » 999999: TAN gültig.
- » 1xxxxx: TAN ok.
- » 2yyyyx: TAN-Rückmeldung mit individuellem Statuscode yyyy.

Wir schauen uns nun an, wie sich einige dieser Fehler auswirken und wie Sie diese abfangen können. Die zuvor vorgestellten Beispiele für verschiedene Geschäftsvorfälle wurden nur mit einer rudimentären Fehlerbehandlung ausgestattet, die nicht alle möglichen Fehler berücksichtigt. Daher finden Sie nachfolgend einige Anregungen, wie Sie Ihre Anwendung um eine geeignete Fehlerbehandlung erweitern können.

# 12.2 Fehlerbehandlungen

Als Beispielformular verwenden wir eine Kopie des Formulars *frmSEPAUeberweisung*, das wir als *frmSEPAUeberweisungFehlerbehandlung* kopiert haben.

Die dort aufgerufene Funktion *SEPAUeberweisung* haben wir gleich in das Klassenmodul des Formulars *frmSEPAUeberweisungFehlerbehandlung* kopiert und *SEPAUeberweisungFehlerbehandlung* genannt.

Wir schauen uns nun an, wie sich die unterschiedlichen Fehler durch die oben genannten PINund TAN-Nummern auswirken und wie Sie diese behandeln können.

## 12.2.1 PIN gesperrt

Wenn Sie die Überweisungsdaten im Formular eingegeben haben und den Fehler einer gesperrten PIN hervorrufen möchten, nehmen Sie am besten eine Überweisung von einem Testkonto zum anderen vor (sonst gelingt dieser Fehler natürlich nicht durch die Eingabe der PIN 01111). Was geschieht hier, wenn wir den Fehler nicht behandeln? Es taucht der Fehler aus Abbildung 12.1 auf. Leseprobe! Vollständiges Buch erhältlich unter www.amvshop.de im André Minhorst Verlag.

Fehlerbehandlungen



Abbildung 12.1: Fehlermeldung bei gesperrtem PIN

Dieser Fehler tritt in der folgenden Zeile auf:

Set objMessage = objBACDialog.ExecuteSegment(objSegment)

Der Fehler wird ausgelöst, weil die Zeile versucht, ein Segment auszuführen, obwohl das *obj-BACDialog*-Objekt nicht gefüllt wurde. Was mit dem *BACDialog*-Objekt los ist, können wir mit der Eigenschaft *State* ermitteln:

Debug.Print objBACDialog.State

Dies liefert einen der folgenden Werte, in diesem Fall den Wert 1:

- » Const bacDialogAborted = 32
- » Const bacDialogBeginning = 2
- » Const bacDialogEnding = 16
- » Const bacDialogExecuting = 8
- » Const bacDialogIdle = 1
- » Const bacDialogReady = 4

*bacDialogIdle* bedeutet, dass der Dialog inaktiv ist. Den Fehler hätten wir weiter oben bereits abfangen können, als das *BACDialog*-Objekt mit der Methode *BeginDialog* gestartet wurde:

```
....
Set objBACDialog = objContact.NewDialogUI
objBACDialog.BeginDialog
Select Case Left(objBACDialog.HBCIResultCode, 1)
    Case "9"
    MsgBox "Fehler " & objBACDialog.HBCIResultCode & vbCrLf & vbCrLf _
    & objBACDialog.HBCIResult
```

### Kapitel 12 Fehlerbehandlung

```
Exit Function
Case Else
Debug.Print objBACDialog.HBCIResultCode, objBACDialog.HBCIResult
End Select
```

Hier prüfen wir nun die erste Stelle der Eigenschaft *HBCIResultCode*. Ist dieser *9*, ist ein Fehler aufgetreten. Geben Sie hier also den PIN 01111 ein, erscheint nun die Meldung aus Abbildung 12.2 und die Funktion wird beendet.

-8	Überweisung		×	
SEPA-Überweisung/Terminüberweisung				
Bankverbindung:	ankverbindung: Datadesign Demobank 160 70000997 735160			
Konto/Depot:	1000735160 Girokonto			
IBAN:	DE43700009971000735160			
BIC:	DDBADEMM002	Microsoft Access		
Empfänger:				
Empfänger:	DataDesign Demobank 735161	Fehler 9931	1	
Bezeichnung:	DataDesign Demobank 735161	Tallacharanistananat		
Empfängername 1:	DataDesign Demobank 7:	rennenmer ist gesperit.		
Empfängername 2:				
IBAN:	DE16700009971000735161	ОК		
BIC:	DDBADEMM002			
Sonstige Daten:				

Abbildung 12.2: Meldung bei einer gesperrten PIN

### 12.2.2 PIN muss vor erstmaliger Anmeldung geändert werden

Um diesen Fehler zu testen, geben Sie als PIN die Nummer 02222 an. In diesem Fall tritt der Fehler viel weiter hinten in der folgenden Zeile auf, also beim Zugriff auf die Eigenschaft Acknowledgement des ersten Eintrags der Transactions-Auflistung:

```
objMessage.Transactions(0).Acknowledgement.SaveAs _
CurrentProject.Path & "\tracknowledgement.txt"
```

Die vorherige Ausgabe von HBCIResult und HBCIResultCode im Direktfenster liefert dies:

3060 Teilweise liegen Warnungen/Hinweise vor.

Das hilft auch nicht viel weiter, also untersuchen wir weiter. Was liefert die Acknowledgement-Eigenschaft? Darüber geben die folgenden drei Zeilen Auskunft, unmittelbar vor der fehlerhaften Zeile platziert:

### Fehlerbehandlungen

```
Dim objAck As BACSegment
Set objAck = objMessage.Acknowledgement
Debug.Print FormatXML(objAck.GetXML)
```

### Das Ergebnis liefert den Fehler 9050, aber auch ohne genauere Informationen:

```
<?xml version="1.0" encoding="UTF-16"?>

<SEGMENT TYPE="HIRMG" VERSION="2" SEQNO="2">

<Rueckmeldung>

<Code Format="digits">9050</Code>

<Text Format="alphanumeric">Nachricht teilweise fehlerhaft.</Text>

</Rueckmeldung>

</SEGMENT>
```

Immerhin aber erhalten wir zumindest zwei *Transaction*-Objekte, wie die entsprechende Ausgabe beim Debuggen der entsprechenden Zeile im Direktfenster offenbart:

```
? objMessage.Transactions.Count
2
```

### Das erste Transaction-Element liefert aber schon mal keine Antwort-Segmente:

```
? objMessage.Transactions(0).ResponseSegments.Count 0
```

Vielleicht das zweite? Nein, auch dort ist die Anzahl *O*. Also schauen wir die Eigenschaft *Acknowledgement* für beide *Transaction*-Elemente an: Das *Acknowledgement*-Element für die erste *Transaction* ist leer, das zweite auch. Auch die *OrderSegment*-Elemente liefern keine weiteren Informationen. Wie es aussieht, können wir hier keine spezifischeren Informationen herausfinden und müssen eine allgemeine Fehlermeldung ausgeben.

Immerhin wissen wir, wodurch der Fehler ausgelöst wurde – nämlich durch den Aufruf der *ExecuteSegment*-Anweisung. Also fügen wir neben der ersten Fehlerbehandlung noch eine zweite hinter dieser Anweisung hinzu – sodass beide Fehlerbehandlungen nun so aussehen:

### Kapitel 12 Fehlerbehandlung

```
Case "9"

MsgBox "Fehler beim Ausführen des Segments: "________

& objBACDialog.HBCIResultCode & vbCrLf & vbCrLf & objBACDialog.HBCIResult

Exit Function

Case Else

Debug.Print objBACDialog.HBCIResultCode, objBACDialog.HBCIResult

End Select
```

## 12.2.3 Abbruch des Dialogs und Rückmeldung mit Code 9333

Geben Sie den PIN *03333* ein, löst dies den Fehler aus Abbildung 12.3 aus. Immerhin mal ein Fehler, den wir durch den Einbau unserer beiden Fehlerbehandlungen abfangen konnten – in diesem Fall durch die Fehlerbehandlung, die gleich auf die *BeginDialog*-Methode folgt.



Abbildung 12.3: Fehler bei Eingabe von PIN 03333

## 12.2.4 Vorläufige PIN-Sperre

Mit dem Code *99998* für die PIN erzeugen wir eine vorläufige PIN-Sperre. Hier taucht die Fehlermeldung aus Abbildung 12.4 auf. Auch dieser Fehlermeldung ist nicht sonderlich aussagekräftig.

Microsoft Access			
Fehler beim Ausführen des Segments: 9050			
Nachricht teilweise fehlerhaft.			
ОК			

Abbildung 12.4: Fehlermeldung bei einer PIN-Sperre

# 13 Tools und Hilfsfunktionen

Dieses Kapitel liefert die in der Beispieldatenbank verwendten Tools und Hilfsfunktionen.

## 13.1 Die Funktion GetXMLElement

Diese Funktion durchsucht das mit dem ersten Parameter übergebenen XML-Dokuments nach einer mit dem zweiten Parameter gelieferten Suchausdruck. Dieser muss der *XPath*-Syntax entsprechen, einer speziellen Notation für die Suche in XML-Dokumenten. In den vorliegenden Fällen hangeln wir uns dazu einfach an der Hierarchie der entsprechenden Elemente entlang, beispielsweise mit //SEGMENT/Parameter/MinVorlaufzeit. Die Funktion verwendet schließlich die *selectSingleNode*-Funktion, um das im Suchpfad angegebene Element zu finden. Wurde das Element gefunden und mit *objNode* referenziert, landet der enthaltene Text im Rückgabewert *GetXMLElement*:

```
Public Function GetXMLElement(strXML As String, strElement As String)
Dim objXML As MSXML2.DOMDocument
Dim objElement As MSXML2.IXMLDOMNode
Dim objNode As MSXML2.IXMLDOMNode
Set objXML = New MSXML2.DOMDocument
objXML.LoadXML strXML
Set objElement = objXML.documentElement
Set objNode = objElement.selectSingleNode(strElement)
If Not objNode Is Nothing Then
GetXMLElement = objNode.nodeTypedValue
End If
End Function
```

# 13.2 Die Funktion FormatXML

Diese Funktion erwartet ein XML-Dokument als *String*-Parameter. Sie fügt dem üblicherweise als einzeiligen String gelieferten Inhalt die notwendigen Zeilenumbrüche für eine bessere Lesbarkeit hinzu und gibt das formatierte XML-Dokument als Funktionswert zurück:

```
Public Function FormatXML(strXML As String) As String
Dim strTemp As String
Dim str() As String
Dim i As Integer
Dim intSlash As Integer
```

#### Kapitel 13 Tools und Hilfsfunktionen

```
Dim intEinruecken As Integer
   Dim bolDescription As Boolean
    strTemp = strXML
    strTemp = Replace(strTemp, "><", ">|~|<")</pre>
    str = Split(strTemp, "|~|")
   For i = LBound(str) To UBound(str)
        If str(i) = "<Description>" Then
            bolDescription = True
        ElseIf str(i) = "</Description>" Then
            bolDescription = False
        End If
        If Not bolDescription Then
            intSlash = InStr(1, str(i), "</")</pre>
            If intSlash = 1 Then
                intFinruecken = intFinruecken - 4
            End If
            If intEinruecken < 0 Then intEinruecken = 0
            str(i) = Space(intEinruecken) & str(i)
            If intSlash = 0 Then
                intSlash = InStr(1, str(i), "/>")
                If intSlash = 0 Then
                    intEinruecken = intEinruecken + 4
                End If
           Fnd If
       End If
   Next i
   FormatXML = Join(str, vbCrLf)
End Function
```

## 13.3 Die Funktion IsoDatum

Die Funktion *IsoDatum* wandelt Datum und Uhrzeit so um, dass es in SQL-Ausdrücken verwendet werden kann:

```
Public Function Isodatum(datDatum As Date) As String
Isodatum = Format(datDatum, "\#yyyy\-mm\-dd\ hh:nn:ss#")
End Function
```

Ein Beispiel sieht wie folgt aus:

```
? IsoDatum("26.09.2014 13:35:14")
#2014-09-26 13:35:14#
```

# 14 Onlinebanking per Webservice

Das aktuellste Angebot der Firma *B+S Banksysteme AG* ist ein Webservice, über den Sie prinzipiell alle Aktionen, die Sie auch über die DDBAC-Komponenten durchführen können, aufrufen. Das liefert folgende Vorteile:

- » Sie müssen neben der Access-Anwendung, die Sie mit Onlinebanking-Funktionen ausstatten möchten, keine weiteren Komponenten für das Onlinebanking installieren.
- » Sollten einmal Änderungen an den Komponenten für das Onlinebanking nötig sein, wird dies direkt auf dem Server erledigt. Sie müssen also keine Aktualisierung der Software auf den betroffenen Rechnern vornehmen.
- » Die Programmierung des Webservice ist sehr einfach gehalten.

Durch diese Einfachheit gibt es allerdings auch ein paar Nachteile, die wie folgt aussehen:

Es werden nicht alle Geschäftsvorfälle unterstützt, sondern nur einige wenige – Kontostand einlesen, Umsätze ermitteln und Überweisungen (gegebenenfalls mit Termin) tätigen. Dies sind allerdings auch die Vorgänge, mit denen die meisten Nutzer auskommen dürften.

Ein weiterer Nachteil ist, dass nur PIN/TAN als Sicherheitsverfahren unterstützt wird. Dies schließt die Benutzer einiger Bankinstitute aus, da nicht alle Banken dieses Verfahren anbieten. Sie können bei solchen Banken wie beispielsweise bei der Commerzbank zwar online mit PIN und TAN arbeiten, aber nicht mit HBCI. Die Nutzung des Webservice ist also somit Benutzern vorbehalten, die eine Lösung für sich selbst bauen und wissen, dass die Kreditinstitute, die betroffen sind, PIN/TAN per HBCI anbieten.

## 14.1 Funktionsweise des Webservice

Die Funktionsweise ist einfach: Sie verwenden einige Zeilen Code, um eine SOAP-Anfrage zusammenzustellen. Diese sieht, wenn Sie etwa die Informationen zu einer Bank mit einer bestimmten Bankleitzahl abrufen möchten, etwa wie folgt aus:

### Leseprobe! Vollständiges Buch erhältlich unter www.amvshop.de im André Minhorst Verlag.

### Kapitel 14 Onlinebanking per Webservice

Diese Anfrage ruft beispielsweise die Informationen zur Bank mit der Bankleitzahl 350 400 38 ab und liefert diese ebenfalls in Form einer SOAP-Antwort zurück. Diese sieht, in Auszügen, beispielsweise wie folgt aus:

```
<?xml version="1.0"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"...>
<soap:Body>
<GetBankInfoResponse xmlns="http://service.ddbac.de/">
<GetBankInfoResponse xmlns="http://service.ddbac.de/">
<GetBankInfoResult>
<BankCode>35040038</BankCode>
<BIC>COBADEFF350</BIC>
<Bankname>Commerzbank</Bankname>
....
</GetBankInfoResult>
</GetBankInfoResponse>
</soap:Body>
</soap:Envelope>
```

Sowohl die im Request also auch im Response enthaltenen Informationen liegen im XML-Format vor, auf das wir mithilfe der Bibliothek *Microsoft XML, v6.0* leicht zugreifen können. Dazu fügen Sie zunächst einen Verweis auf die entsprechende Bibliothek hinzu. Öffnen Sie den *Verweise*-Dialog vom VBA-Editor aus mit dem Menübefehl *Extras | Verweise* und markieren Sie den Eintrag aus Abbildung 14.1.

Verweise - prjOnlinebanking	×
Verfügbare Verweise 1:	ОК
✓ Visual Basic For Applications	Abbrechen
CUE Automation CMicrosoft Office 14.0 Access database engine Object Catabasian DDBAC HBCI Banking Application Compor	Durchsuchen
Microsoft XML, v6.0     APSBFC0     Microsoft ActiveX Data Objects 2.5 Library     AccessibilityCplAdmin 1.0 Type Library     Acrobat     Acrobat Access 3.0 Type Library     Acrobat Scal 1.0 Type Library	Hilfe
Microsoft XML, v6.0 Pfad: C:\WINDOWS\\$YSTEM32\MSXML6.DLL Sprache: Voreinstellung	

Abbildung 14.1: Hinzufügen eines Verweises auf die Bibliothek Microsoft XML, v6.0

SOAP-Request ermitteln/erstellen

# 14.2 SOAP-Request ermitteln/erstellen

Das Erstellen eines SOAP-Requests geschieht meist so: Man sucht sich im Internet irgendein Beispiel für das Absenden eines SOAP-Requests und passt diesen dann auf die eigenen Bedürfnisse an. Das Anpassen ist meist einfach, da die Dokumentation des Webservice in der Regel ein Beispiel für den SOAP-Request für eine der grundlegenden Methoden der Webservice-API liefert. Sprich: XML-Code mit dem SOAP-Request in eine bestehende VBA-Lösung integrieren, Parameter anpassen, Auswertung anpassen – fertig!

### 14.2.1 www.soapclient.com

Wenn es jedoch einmal kein Beispiel für den SOAP-Request auf der Seite des Herstellers gibt, steht man auf dem Schlauch – zumindest ohne die Hilfe fleißiger Helferlein im Internet.

In diesem Fall habe ich schnell http://www.soapclient.com/soaptest.html gefunden. Dort geben Sie die URL der WSDL-Datei des Webservice an. WSDL (Web Service Description Language) ist eine Sprache, mit der die Beschreibungen von Webservices verfasst werden. Die WSDL-Datei enthält alle Methoden und Elemente eines Webservices.

Wenn Sie diese Datei im Webbrowser anzeigen lassen, sieht das etwa wie in Abbildung 14.2 aus. In diesem Fall sehen Sie die WSDL-Datei für den DDBAC-Webservice, die Sie unter folgendem Link finden:



https://service.ddbac.de/demo/ServiceNG.asmx/WDSL

Abbildung 14.2: Beispiel für eine WSDL-Datei

Nun geben Sie diesen Link auf der Seite *http://www.soapclient.com/soaptest.html* ein (siehe Abbildung 14.3). Klicken Sie dann auf *Retrieve*.

### Leseprobe! Vollständiges Buch erhältlich unter www.amvshop.de im André Minhorst Verlag.

Kapitel 14 Onlinebanking per Webservice



Abbildung 14.3: Eingabe der WSDL-Datei in einen Service, der die Syntax für einen Request liefern soll

Nach diesem Schritt zeigt der Soapclient die Methoden des Webservices samt allen verfügbaren Parametern an (siehe Abbildung 14.4). Hier wählen wir beispielsweise gleich die erste Methode namens *GetBankInfo* aus und geben für den Parameter *sBankCode* die Bankleitzahl ein (zum Beispiel *350 400 38*). Wir wollen zunächst wissen, wie der Request aussieht, denn diesen müssen wir ja per Code zusammenstellen und an den Webservice schicken. Also wählen Sie unter Show den Wert *Request* aus und klicken auf die Schaltfläche *Invoke*.
#### SOAP-Request ermitteln/erstellen

Pearlaciton Ansight Environten Extras 2		×	Konvertieren
beabeten Ansient Pavonten Extras .		^	Anonventeren
50APClient			
Home   SOAP Tools   UDDI Browser	Resources   Source Code   RFCs	News Reader   SOAP Interop   Bookmar	ks
	ServiceNG	None	
	Service Documentation		
	SOAP Method : GetBank	Info	
ServerAddress:	https://service.ddbac.de/ddbac-dev-	1/ServiceNG.asmx	
GetBankInfo.sBankCode[0]:			
Show: Response 🗸	Format: XML 🗸	Invoke	
SOA	P Method : CreateContactDir	rectRequest	
ServerAddress	https://service.ddbac.de/ddbac-dev-	·1/ServiceNG.asmx	
CreateContactDirectRequest.CustomerData[0]	.BankCode[0]:		
CreateContactDirectRequest.CustomerData[0]	.UserId[0]:		_
CreateContactDirectRequest.CustomerData[0]	.CustomerId[0]:		_
CreateContactDirectRequest.Pin[0]:			_

Abbildung 14.4: Eingabe der Parameter für die gewünschte Methode

Das Ergebnis überzeugt: Der Webbrowser zeigt gleich im Anschluss die komplette SOAP-Datei an (siehe Abbildung 14.5).

← (⇒) ﷺ http://www.soapclient.com/S タ マ ♂) ﷺ soapclient.com ×	Ĥ ★ 🕸
Datei Bearbeiten Ansicht Favoriten Extras ?	🗶 🍓 Konvertieren 🔻 🛃 Auswähle
<pre><?xml version="1.0" encoding="UTF-8"?> &lt; SOAP-ENV:Envelope xmlns:s0="http://service.ddbac.de/" xmlns:tm=" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:thp: xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" xmlns: xmlns:soap12="http://www.w3.org/2001/XMLSchema" xmlns:soap12/" xmlns: xmlns:SoAP-ENV:Http://www.w3.org/2003/05/soap-envelope"&gt; - <soap-env:body> - <soap-env:body> - <so:getbankinfo xmlns:s0="http://service.ddbac.de/">  </so:getbankinfo></soap-env:body></soap-env:body></pre>	'http://microsoft.com/wsdl/mime/textMatching/" ="http://schemas.xmlsoap.org/wsdl/http/" tp://schemas.xmlsoap.org/soap/encoding/" oap="http://schemas.xmlsoap.org/wsdl/soap/" ://www.w3.org/2001/XMLSchema-instance"

Abbildung 14.5: Der Request zum Abfragen der Detailinformationen eines Kreditinstituts

Perfekt – nun müssen wir nur noch den VBA-Code um diese Methode herum aufbauen, der die einzelnen Elemente zusammenstellt und den oder die Parameter einfügt.

Dummerweise können Sie den XML-Code nicht direkt aus dem Webbrowser herauskopieren, da dieser dann die Minus-Zeichen zum Ein- und Ausblenden der einzelnen Bereiche mitkopiert. Das ist aber kein Problem, denn Sie können beispielsweise im Internet Explorer 11 über den Menübefehl *Ansicht/Quelle* den Quelltext der Datei anzeigen lassen (siehe Abbildung 14.6).



Abbildung 14.6: Quellcode des Requests

### 14.2.2 SoapUI

Eine professionelle Alternative ist das Tool *SoapUI*. Damit können Sie sehr komfortabel auf die in der *WDSL*-Datei gespeicherten Funktionen zugreifen. Nach dem Start des Tools legen Sie über *File*/*New SOAP-Project* ein neues Projekt an. Mit dem nun erscheinenden Dialog geben Sie die Internetadresse der WDSL-Datei des Webservice an (siehe Abbildung 14.7).

SOAP-Request ermitteln/erstellen

A C		
File Teels Deskten Hele		
	Caracter Frances	
	Search Forum	
Projects	SoapUI Starter Page	
December	Constituent interface and the annual second se	
Properties	soaporiog nicpiog jerryiog erroriog wsimiog memoryiog	

Abbildung 14.7: Eingabe der WDSL-Datei für das SoapUI-Projekt

Nach dem Erstellen des neuen Projekts zeigt *SoapUI* im linken Bereich eine Übersicht aller verfügbaren Requests an (siehe Abbildung 14.8).





Abbildung 14.8: Anzeige des Dokuments zum Absenden eines Requests

Wenn Sie nun statt der Fragezeichen (?) als Platzhalter die tatsächlichen Daten für den Request einsetzen, in diesem Fall *BankCode*, *UserID*, *AccountID* und *Pin*, können Sie den Request mit einem Klick auf den Pfeil links oben absenden (siehe Abbildung 14.9).

SOAP-Request ermitteln/erstellen



Abbildung 14.9: Eingabe der Daten für den Request

Das Ergebnis, also die Response, zeigt *SoapUI* nun im rechten Teil des Fensters an (siehe Abbildung 14.10). In diesem Fall haben wir den Request *CreateContactDirectRequest* abgesetzt. Dieser liefert alle Konten für die angegebene *UserId* sowie zwei weitere sehr wichtige Informationen: und zwar eine *AccountID* für jedes Konto und einen Wert namens *ContactData*. Beide benötigen Sie, wenn Sie im weiteren Verlauf Anfragen absetzen möchten – wie beispielsweise zur Ermittlung des Kontostands, der Umsätze oder um eine Überweisung zu tätigen. Darauf kommen wir jedoch später zurück.

Kapitel 14 Onlinebanking per Webservice



Abbildung 14.10: Erhalt der Response vom Webserver

An dieser Stelle ist es wichtig, ein Tool zu kennen, mit dem Sie den Aufbau der Requests und der Responses einsehen können. Damit fällt es erheblich leichter, den Code zum Absetzen der Requests und zum Einlesen der Ergebnisse zu programmieren.

Mit SoapUI können Sie sich schließlich auch die übrigen Requests ansehen.

# 14.3 Mögliche Anfragen per Webservice

Diese Anfragen sehen wir uns nun an:

- » BalanceRequest: Ermittelt den Kontostand für ein Konto. Erwartet die Parameter ContactData und AccountId, die zuvor mit CreateContactDirectRequest erfasst werden müssen.
- » CheckIban: Prüft eine IBAN-Nummer.
- » CreateContactDirectRequest: Liefert alle Daten zu einem Kontakt, also alle Konten, zu jedem Konto eine AccountId sowie einen Wert für ContactData, mit dem weitere Aktionen authentifiziert werden können.
- » GetBankInfo: Liefert Informationen zu einer Bank
- » MoneyTransferRequest: Durchführen einer Überweisung

#### Mögliche Anfragen per Webservice

- » MoneyTransferComplete: Übermittlung der TAN bei einer Überweisung
- » RetrieveBicForBankCode: Ermittlung der BIC aus einer BLZ
- » RetrieveBicForIban: Ermittlung einer BIC aus einer IBAN
- » RetrievelbanAndBic: Ermittlung von IBAN und BIC aus Kontonummer und BLZ
- » StatementRequest: Abrufen des Kontostands

Wir beginnen mit den Abfragen, für die keine Authentifizierung erforderlich ist, also den einfachen Abfragen von BIC und/oder IBAN sowie den Bankinformationen. Dazu haben wir ein Formular vorbereitet, dass alle Funktionen vorstellt (siehe Abbildung 14.11).

	frmWSFunktionen	– 🗆 ×
Webservice	-Funktionen	
BIC aus BLZ		
BLZ:		BIC:
35040038	🔿 BIC für BLZ ermitteln	COBADEFFXXX
BIC und IBAN aus BLZ und Ko	ntonummer	
BLZ:		BIC:
35040038	<ul> <li>BIC upd IBAN aus BIZ</li> </ul>	COBADEFFXXX
Kontonummer:		IBAN:
4151098		DE24350400380415109800
BIC aus IBAN		BIC:
DE24350400380415109800	🚽 🔿 BIC für IBAN ermitteln	COBADEFFXXX
BAN prüfen		
IBAN:		
DE24350400380415109800	IBAN prüfen	🔽 IBAN okay
Bank-Informationen anzeige	n	
BLZ:		
35040038	Bank-Informationen les	sen
📀 ок		

### Abbildung 14.11: Formular zum Ausprobieren aller einfachen Webservice-Funktionen

### Eine Online-Dokumentation zum Webservice finden Sie unter folgendem Link:

https://service.ddbac.de/ddbac-dev-1/ServiceNG/Index.html

### Oder Sie laden die Dokumentation hier herunter:

https://service.ddbac.de/ddbac-dev-1/ServiceNG/ServiceNG.chm

# 14.4 BIC für eine BLZ ermitteln

Anhand der Ermittlung der BIC für eine BLZ schauen wir uns die grundlegende Vorgehensweise für die folgenden Geschäftsvorfälle an. Im Formular *frmWSFunktionen* entspricht dies der oberen Funktion (siehe Abbildung 14.12).

-8	frmWSFunktionen	-		×
	1	· 13 ·	1 14	•
- - 1 -	✓ Detailbereich			
-	BIC aus BLZ		ī	
1	BIZ: BIC: BIC: BIC für BLZ ermitteln Ungebunden			
2	BIC und JBAN aus BLZ und Kontonymmer	-		

Abbildung 14.12: Steuerelemente zum Testen der Funktion RetrieveBicForBankCode

### 14.4.1 Ermittlung per VBA-Funktion

Das linke Textfeld namens *txtBLZZuBIC* erwartet die Angabe der Bankleitzahl, zu der die BIC ermittelt werden soll. Das rechte Textfeld heißt *txtBICAusBLZ* und gibt das Ergebnis aus. Die Schaltfläche *cmdBICAusBLZ* ermittelt bei Anklicken die BIC per Abfrage des Webservices. Die dadurch ausgelöste Prozedur sieht so aus:

```
Private Sub cmdBICAusBLZ Click()
   Dim strBIC As String
   Dim strErrorCode As String
   Dim strErrorText As String
   If Not IsNull(Me!txtBLZZuBIC) Then
       strBIC = RetrieveBICForBankCode(Me!txtBLZZuBIC, strErrorCode, strErrorText)
       If Len(strBIC) > 0 Then
           Me!txtBICAusBLZ = strBIC
       F1se
           MsgBox "BIC konnte nicht ermittelt werden." & vbCrLf & strErrorCode
               & vbCrLf & strErrorText
       End If
   Else
       MsgBox "Geben Sie eine BLZ ein."
       Me!txtBLZZuBIC.SetFocus
   Fnd If
```

BIC für eine BLZ ermitteln

End Sub

Die Prozedur prüft zunächst, ob das Textfeld *txtBLZZuBIC* überhaupt einen Wert enthält. Falls nicht, erscheint eine entsprechende Meldung. Anderenfalls ruft die Prozedur die Funktion *Re-trieveBICForBankCode* auf und übergibt dieser drei Parameter. Der erste enthält die Bankleitzahl aus dem Textfeld, die übrigen beiden nehmen Variablen auf, die in der Prozedur deklariert und von der aufgerufenen Funktion gefüllt werden sollen, falls beim Abfragen der BIC ein Fehler auftritt.

Als Funktionswert gibt die Funktion die BIC zurück – allerdings nur, wenn kein Fehler aufgetreten ist. Wenn die BIC zurückgegeben wird, landet diese im Textfeld *txtBICAusBLZ*, anderenfalls enthalten die beiden Parameter *strErrorCode* und *strErrorText* entsprechende Fehlerinformationen, die per *MsgBox* ausgegeben werden.

# 14.4.2 Funktion zum Abfragen des BIC zu einer BLZ

Die Funktion *RetrieveBICForBankCode* sieht wie folgt aus:

```
Public Function RetrieveBICForBankCode(strBankCode As String,
       strErrorCode As String, strErrorText As String) As String
   Dim objXMLResponse As MSXML2.DOMDocument
   Dim strRequest As String
   Dim strResponse As String
   Dim strFunction As String
   strFunction = "RetrieveBicForBankCode"
   strRequest = CreateSoapRequest("
                                               <ser:sBankCode>" & strBankCode
       & "</ser:sBankCode>". strFunction)
   Request strRequest, objXMLResponse
   strResponse = objXMLResponse.selectSingleNode("//" & strFunction & "Response").XML
   strErrorCode = GetXMLElement(strResponse, strFunction & "Result/ErrorCode")
   strErrorText = GetXMLElement(strResponse, strFunction & "Result/Error/ErrorText")
   RetrieveBICForBankCode = GetXMLElement(strResponse, strFunction & "Result/Bic")
End Function
```

Sie erwartet die von der aufrufenden Prozedur übergebenen Werte beziehungsweise Variablen und hat etwas wenig Zeilen, wenn man bedenkt, dass sie einen Webservice kontaktieren und das Ergebnis auswerten soll. Das liegt daran, dass einige Anweisungen, die in diesen und den anderen noch vorgestellten Aufrufen des Webservice gleich sind, in eigene Funktionen ausgelagert wurden. Diese Funktion schreibt zunächst nur den Namen der zu verwenden API-Funktion des Webservice in die Variable *strFunction*, und zwar *RetrieveBicForBankCode*.

Achtung: Beachten Sie hier wie auch beim Zusammenstellen der XML-Requests penibel genau die Groß- und Kleinschreibung – der Webservice achtet nämlich ebenso genau auf korrekte Bezeichnungen und liefert selbst beim kleinsten Fehler nicht mehr das gewünschte Ergebnis!

#### Kapitel 14 Onlinebanking per Webservice

Die folgende Anweisung stellt schon den Teil des XML-Requests zusammen, der diesen Request von den übrigen Requests unterscheidet. Dann wird die Zeile an die Funktion CreateSoapRequest übergeben, welche die weiteren Zeilen hinzufügt. Die übergebene Zeile sieht wie folgt aus, wenn die Variablen gefüllt sind:

```
<ser:sBankCode>xxxxxx</ser:sBankCode>
```

Sie enthält also schlicht und einfach ein Element mit der Bankleitzahl. Die für die weitere Verarbeitung verwendete Funktion *CreateSoapRequest* sieht wie folgt aus:

Sie erwartet die Basiszeile sowie den Namen der API-Funktion des Webservice als Parameter und ruft einige weitere Funktionen auf, welche jeweils einen Teil des Requests zusammenstellen. Die erste Funktion heißt *RequestEnvelopeStart*:

### Sie legt die erste Zeile des Requests an:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="http://service.ddbac.de/">
```

# Die zweite Funktion namens *RequestBodyStart* sieht so aus und erwartet den Funktionsnamen aus *strFunction* als Parameter:

```
Public Function RequestBodyStart(strMethod As String) As String
Dim strRequest As String
strRequest = strRequest & " <soapenv:Body>" & vbCrLf
strRequest = strRequest & " <ser:" & strMethod & ">" & vbCrLf
RequestBodyStart = strRequest
```

### BIC für eine BLZ ermitteln

End Function

### Sie fügt die folgenden Zeilen hinzu und fügt dabei den Funktionsnamen ein:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"7
```

xmlns:ser="http://service.ddbac.de/">

<soapenv:Body>

<ser:RetrieveBicForBankCode>

### Dann folgt der bereits in der aufrufenden Routine zusammengestellte Teil mit den zu übergebenden Parametern:

<ser:sBankCode>35040038</ser:sBankCode>

### Die folgende Funktion sieht so aus:

```
Public Function RequestBodyEnde(strMethod As String) As String
Dim strRequest As String
strRequest = strRequest & " </ser:" & strMethod & ">" & vbCrLf
strRequest = strRequest & " </soapenv:Body>" & vbCrLf
RequestBodyEnde = strRequest
End Function
```

### Sie fügt weitere zwei Zeilen hinzu:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" 7
```

xmlns:ser="http://service.ddbac.de/">

<soapenv:Body>
 <ser:RetrieveBicForBankCode>
 <ser:sBankCode>35040038</ser:sBankCode>
 </ser:RetrieveBicForBankCode>

</soapenv:Body>

### Schließlich folgt noch die letzte Funktion:

```
Public Function RequestEnvelopeEnde() As String
    RequestEnvelopeEnde = "</soapenv:Envelope>" & vbCrLf
End Function
```

### Der vollständige Request sieht nun so aus:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" 7</pre>
```

xmlns:ser="http://service.ddbac.de/">

```
<soapenv:Body>
<ser:RetrieveBicForBankCode>
<ser:sBankCode>35040038</ser:sBankCode>
</ser:RetrieveBicForBankCode>
</soapenv:Body>
</soapenv:Envelope>
```

Nachdem der Request mit diesen Hilfsfunktionen zusammengestellt wurde, übergibt die Routine *RetrieveBICForBankCode* den Request an die Funktion *Request*. Diese sieht so aus und erwartet den Request sowie eine Variable für das *Response*-Objekt als Parameter:

```
Public Function Request(strRequest As String, 7
       objXMLResponse As MSXML2.DOMDocument) As Boolean
   Dim strURL As String
   Dim objXMLHTTP As MSXML2.XMLHTTP60
   Set objXMLHTTP = New MSXML2.XMLHTTP60
   Set objXMLResponse = New MSXML2.DOMDocument
   strURL = cURLBase
   With objXMLHTTP
       .Open "post", strURL, False
        .setRequestHeader "Content-Type", "text/xml; charset=utf-8"
       .setReguestHeader "Content-Length", Len(strReguest)
        .send strRequest
       objXMLResponse.LoadXML .responseText
       Request = True
   End With
End Function
```

Die Funktion erfragt die Basis-URL aus der Konstanten *cURLBase*, die wir wie folgt im Kopf des Moduls hinterlegt haben:

```
Public Const cURLBase As String = "https://service.ddbac.de/demo/ServiceNG.asmx"
```

Damit erstellt sie ein neues Objekt des Typs *XMLHTTP60* und übergibt der *Open*-Methode dieses Objekts den Parameter *post*, die URL sowie den Wert *False*. Der erste gibt an, dass die *POST*-Methode für die Anfrage verwendet werden soll, der dritte, dass der Request nicht asynchron erfolgen soll. Die aufrufende Prozedur wird somit solange angehalten, bis der Webservice das Ergebnis zurückgeliefert hat. Nach dem Setzen zweier Header, von denen der erste den Zeichensatz festlegt und der zweite die Zeichenanzahl des Request übergibt, folgt der Aufruf der *send*-Methode. Diese schickt die Anfrage an den Webservice.

Die *responseText*-Eigenschaft des *XMLHTTP60*-Objekts liefert das Ergebnis als Text, der dann in einem Objekt des Typs *DOMDocument* landet, das als Ergebnis an die aufrufende Routine zu-

### BIC für eine BLZ ermitteln

rückgegeben wird. Das im *DOMDocument* gespeicherte Ergebnis sieht beispielsweise wie folgt aus – mehr dazu gleich im Anschluss:

Dieses XML-Dokument wird dann an die aufrufende Funktion *RetrieveBICForBankCode* zurückgegeben. Da wir die *soap:...*-Elemente nicht benötigen, ermittelt die Funktion zunächst mit der *selectSingleNode*-Methode den Teil des Dokuments, der uns interessiert – nämlich das Element *RetrieveBicForBankCodeResponse*. Dies erledigt die Funktion mit der *selectSingleNode*-Funktion, die das gewünschte Element mit der *XPath*-Syntax ermittelt (in diesem Fall liefert // *RetrieveBicForBankCodeResult* genau das gewünschte Element). Der verbleibende Teil sieht nun so aus:

```
<RetrieveBicForBankCodeResponse xmlns="http://service.ddbac.de/">
    <RetrieveBicForBankCodeResult>
        <SuccessText>Erfolgreich</SuccessText>
        <Bic>COBADEFFXXX</Bic>
        </RetrieveBicForBankCodeResult>
    </RetrieveBicForBankCodeResponse>
```

Hier ist zunächst die Erfolgsmeldung aus dem Element *SuccessText* erfreulich. Dennoch versucht die Funktion, mit der Hilfsfunktion *GetXMLElement* eventuell für die Elemente *RetrieveBicForBankCodeResult/Error/ErrorCode* und *RetrieveBicForBankCodeResult/Error/ErrorText* gelieferte Werte einzulesen und speichert diese gegebenenfalls in den beiden Variablen *strErrorCode* und *strErrorText*, die als Parameter an die aufrufende Routine zurückgegeben werden. Der Inhalt des Elements *Bic*, der das gewünschte Ergebnis enthält, wird dann ebenfalls mit *GetXMLElement* ermittelt und als Rückgabewert der Funktion festgelegt.

Schauen wir uns noch an, wie die Antwort aussieht, wenn der Aufruf nicht erfolgreich ist. Dazu übergeben wir eine offensichtlich nicht gültige BLZ an den Webservice. Hier ist das Resultat:

<RetrieveBicForBankCodeResponse xmlns="http://service.ddbac.de/">

```
Kapitel 14 Onlinebanking per Webservice
</retrieveBicForBankCodeResult>
        <Error>
            <ErrorCode>INVALID_BANK_CODE</ErrorCode>
            <ErrorText>Ungültige Bankleitzahl</ErrorText>
            <NoFurtherRequestsPreferred>false</NoFurtherRequestsPreferred>
            </Error>
            </RetrieveBicForBankCodeResult>
            </RetrieveBicForBankCodeResponse>
```

Die hier gelieferten Elemente *Error/ErrorCode* und *Error/ErrorText* landen dann in den dafür vorgesehenen Parametern *strErrorCode* und *strErrorText*. Die Prozedur *cmdBICAusIBAN\_Click* gibt dann eine entsprechende Meldung aus (siehe Abbildung 14.13).



Abbildung 14.13: Fehlermeldung bei Übergabe einer ungültigen BLZ

# 14.5 IBAN und BIC ermitteln

Die zweite Funktion, die wir im Formular *frmWSFunktionen* abgebildet haben, erwartet die Eingabe einer BLZ und der Kontonummer und liefert die passende IBAN und BIC. Die Textfelder *txt-BLZ* und *txtKontonummer* nehmen die zu konvertierenden Daten auf, die Textfelder *txtBIC* und *txtIBAN* liefern das Ergebnis. Dazwischen steht die Schaltfläche *cmdIBANUndBICAusBLZUndKontonummer* (siehe Abbildung 14.14).

### **IBAN und BIC ermitteln**

- 0	frmWSFunktionen	_	□ ×	
1	. 5 . 1 . 6 . 1 . 7 . 1 . 8 . 1 .	9 · · · 10 · · · 11 · · · 12 · · · 13 ·	· I · 14 · 🔺	
Formularkopf				
	Funktionen			
Detailbereich				
BIC aus BLZ			7	
BLZ: Ungebunden	→ BIC für BLZ ermitteln	BIC: Ungebunden		
2 BIC und IBAN aus BLZ und Kontonummer				
3 BLZ:		BIC:		
- Ungebunden 4 Kontonummer:	BIC und IBAN aus BLZ	Ungebunden IBAN:		
- Ungebunden	]	Ungebunden		

Abbildung 14.14: BIC und IBAN aus BLZ und Kontonummer ermitteln

Die Schaltfläche löst die folgende Prozedur aus:

```
Private Sub cmdIBANUndBICAusBLZUndKontonummer Click()
   Dim strBIC As String
   Dim strIban As String
   Dim strErrorCode As String
   Dim strErrorText As String
   If Not IsNull(Me!txtBLZ) And Not IsNull(Me!txtKontonummer) Then
       RetrieveIbanAndBic Me!txtBLZ, Me!txtKontonummer, strBIC, strIban,
            strErrorCode. strErrorText
       If Len(strBIC) > 0 Then
           Me!txtBIC = strBIC
           Me!txtIBAN = strIban
       Else
           MsgBox "BIC und IBAN konnten nicht ermittelt werden."
                & vbCrLf & strErrorCode & vbCrLf & strErrorText
       Fnd If
   Else
       MsgBox "Geben Sie BLZ und Kontonummer ein."
       Me!txtBLZ.SetFocus
   End If
End Sub
```

Die Prozedur prüft, ob *txtBLZ* und *txtKontonummer* gefüllt sind und gibt im Zweifelsfalle eine entsprechende Meldung aus. Liegen die Daten vor, ruft sie die Funktion *RetrievelbanAndBic* auf. Diese soll im Gegensatz zur vorherigen Funktion nicht nur einen Wert zurückliefern, sondern

gleich zwei – nämlich die BIC und die IBAN. Dementsprechend richten wir für beide entsprechende Variablen in der aufrufenden Prozedur ein, die wir leer an die Funktion *RetrievelbanAndBic* übergeben – genauso wie die bereits zuvor verwendeten Variablen *strErrorCode* und *strError Text* und die zu konvertierenden Daten aus den beiden Textfeldern *txtBLZ* und *txtKontonummer*. Wenn beim Zugriff auf den Webservice alles klappt, liefert die Funktion entsprechende Werte für die beiden Parameter *strBIC* und *strIBAN* zurück. Beide schreibt die Prozedur dann in die passenden Textfelder. Gelingt der Aufruf nicht, gibt ein Meldungsfenster eine Fehlermeldung aus.

Die Funktion *RetrieveBicAndlban* sieht wie folgt aus. Hier ist zu beachten, dass die Elemente mit den Ausgangswerten kein führendes *s* enthalten – im vorherigen Beispiel hieß das Element für die Bankleitzahl schließlich noch *sBankCode*. Hier heißen die beiden Elemente schlicht *BankCode* und *AccountNumber*. Halten Sie sich im Zweifel einfach an die mit SoapUI oder einem anderen Werkzeug ermittelten Vorgaben:

```
Public Function RetrieveIbanAndBic(strBLZ As String, strKontonummer As String,
       strBIC As String, strIban As String,
       strErrorCode As String, strErrorText As String)
   Dim objXMLResponse As MSXML2.DOMDocument
   Dim strRequest As String
   Dim strResponse As String
   Dim strFunction As String
   strFunction = "RetrieveIbanAndBic"
   strRequest = "
                             <ser:BankCode>" & strBLZ & "</ser:BankCode>" & vbCrLf
   strRequest = strRequest & "
                                         <ser:AccountNumber>" & strKontonummer
       & "</ser:AccountNumber>"
   strRequest = CreateSoapRequest(strRequest, strFunction)
   Request strRequest, objXMLResponse
   strResponse = objXMLResponse.selectSingleNode("//" & strFunction & "Response").XML
   strErrorCode = GetXMLElement(strResponse, strFunction & "Result/Error/ErrorCode")
   strErrorText = GetXMLElement(strResponse, strFunction & "Result/ErrorText")
   strBIC = GetXMLElement(strResponse, strFunction & "Result/Bic")
   strIban = GetXMLElement(strResponse, strFunction & "Result/Iban")
End Function
```

Die Funktion arbeitet prinzipiell genau wie die aus dem vorherigen Beispiel – mit ein paar kleinen Unterschieden. Der erste ist, dass hier zwei Elemente mit Parametern für die Übergabe an den Webservice gefüllt werden. Dieser Request sieht so aus:

**BIC für IBAN ermitteln** 

```
<ser:AccountNumber>4151098</ser:AccountNumber>
    </ser:RetrieveIbanAndBic>
    </soapenv:Body>
</soapenv:Envelope>
```

# Außerdem liefert der Webservice mit dem Response auch zwei Elemente mit den gewünschten Werten zurück:

```
<RetrieveIbanAndBicResponse xmlns="http://service.ddbac.de/">
    <RetrieveIbanAndBicResult>
        <SuccessText>Erfolgreich</SuccessText>
        <Iban>DE24350400380415109800</Iban>
        <Bic>COBADEFFXXX</Bic>
        </RetrieveIbanAndBicResult>
    </RetrieveIbanAndBicResponse>
```

Diese Werte liest die Funktion *RetrievelbanAndBic* allerdings prinzipiell genauso aus wie im vorherigen Beispiel, sodass sich hier kaum Stolperstricke verstecken, wenn man das Prinzip einmal verstanden hat. Wichtig ist noch, dass mit *strFunction* der Ausdruck *RetrievelbanAndBic* übergeben wird, damit die SOAP-Anfragen entsprechend gefüllt werden.

Die ermittelten Werte liest die Prozedur aus und trägt diese in die Zieltextfelder ein.

# 14.6 BIC für IBAN ermitteln

Das Ermitteln der BIC für eine IBAN erfolgt prinzipiell wiederum genau wie das der BIC aus einer Bankleitzahl. Der Request sieht etwas anders aus und verwendet die Funktion *RetrieveBicForlban*:

### Und hier kommt der Response vom Webserver, wenn kein Fehler auftritt:

```
<RetrieveBicForIbanResponse xmlns="http://service.ddbac.de/">
<RetrieveBicForIbanResult>
<SuccessText>Erfolgreich</SuccessText>
<Bic>COBADEFFXXX</Bic>
</RetrieveBicForIbanResult>
```

</RetrieveBicForIbanResponse>

# 14.7 IBAN prüfen

Die Schaltfläche, die den im Textfeld *txtIBAN* enthaltenen Ausdruck auf Gültigkeit prüft, löst die Prozedur *cmdIBANPruefen\_Click* aus (siehe Abbildung 14.15).

-8	frmWSFunktionen	-		×
TRO WO	ebservice-Funktionen			
Caus BLZ -	and a second		4	a la d
<u>`</u>	/			_
-IBAN prüfen				
IBAN:				
DE24350400	380415109800 🔿 IBAN prüfen 🛛 🕅 IBAN okay			
				_
ОК				

Abbildung 14.15: Prüfen der Gültigkeit einer IBAN

Diese Prozedur wollen wir uns noch anschauen, weil das Ergebnis als *Boolean*-Wert zurückgeliefert und direkt als Wert des Kontrollkästchens *chkIBANOkay* angegeben werden soll. Deshalb wird der erwartete Wert auch gleich als *Boolean*-Variable namens *bollban* deklariert:

```
Private Sub cmdIBANPruefen Click()
   Dim bollban As Boolean
   Dim strErrorCode As String
   Dim strErrorText As String
   If Not IsNull(Me!txtIbanPruefen) Then
       bollban = CheckIban(Me!txtIbanPruefen, strErrorCode, strErrorText)
       If Len(strErrorCode) = 0 Then
           Me!chkIBANOkay = bollban
       Else
           MsgBox "IBAN konnte nicht geprüft werden." & vbCrLf & strErrorCode
               & vbCrLf & strErrorText
       End If
   Else
       MsgBox "Geben Sie einen IBAN ein."
       Me!txtIbanPruefen.SetFocus
   Fnd If
```

#### IBAN prüfen

End Sub

Die Funktion *Checklban* hat dementsprechend den Datentyp *Boolean*. Wie aber wandelt sie den typischerweise als String aus dem Element des Response-Dokuments gelieferten Ausdruck (*true/false*) in einen Wert vom Typ *Boolean* um? Dies erledigt die *Eval*-Funktion: *Eval*("*true*") liefert den Wert -1, *Eval*("*false*") den Wert 0:

```
Public Function CheckIban(strIban As String, strErrorCode As String,
       strErrorText As String) As Boolean
   Dim objXMLResponse As MSXML2.DOMDocument
   Dim strRequest As String
   Dim strResponse As String
   Dim strFunction As String
   strFunction = "CheckIban"
   strRequest = CreateSoapRequest("
                                             <ser:sIban>" & strIban
       & "</ser:sIban>", strFunction)
   Request strRequest. objXMLResponse
   strResponse = objXMLResponse.selectSingleNode("//" & strFunction & "Response").XML
   strErrorCode = GetXMLElement(strResponse, strFunction & "Result/Error/ErrorCode")
   strErrorText = GetXMLElement(strResponse, strFunction & "Result/Error/ErrorText")
   CheckIban = Eval(GetXMLElement(strResponse, strFunction & "Result"))
End Function
```

### Der Request sieht beispielsweise so aus:

</soapenv:Envelope>

### Der Webservice liefert dann diese Antwort:

</soap:Body> </soap:Envelope>

# 14.8 Bankinformationen abfragen

Die letzte Funktion dieses Formulars liefert einen Satz von Bank-Informationen (siehe Abbildung 14.16).

-8	frmWSFunktionen	-		×
<b>IS</b>	Webservice-Funktionen			
-RIC aus			•	
-Bank-In	formationen anzeigen			
BLZ:				
350400	38 Bank-Informationen lesen			
💽 ок				

Abbildung 14.16: Anfordern von Bank-Informationen

### Der Request sieht so aus:

Die Bank liefert ein etwas längliches XML-Dokument zurück, das wir an dieser Stelle nicht abbilden wollen – es enthält auch keine wichtigen Informationen, die für die Durchführung der folgenden Geschäftsvorfälle nötig wären.

# Α

Abholzeitpunkt 219 AbweichendeBetragswaehrung1 344 AbweichenderBetrag1 344 AccountData 67, 105 AccountErstellen 268, 348 AccountID 477 AccountNumber 146 Accounts 69 AccountsEinlesen 87 AcctName 146 Acknowledgement 119, 120, 233, 235, 271, 303, 304, 333, 341 Acknowledgements.txt 289 ActivateTANList 73, 75, 77, 78 Administrator für Homebanking Kontakte 17 Aktualisieren der Konten nach Kontakt-Auswahl 186 Alle BACCustomer löschen 51 AlleKonten 232 AlleKonten1 102, 107, 190 Alle Konten einlesen 192 Amount 303, 331 an(..x) 101 Anzahl1 344 Auf BACAccount-Objekte zugreifen 84 Aufsetzpunkt 232 Aufsetzpunkt1 235 AuftragAenderbar 296 AuftragAenderbar1 344 AuftragAussetzbar1 344 AuftraggeberID 242 AuftraggeberKontoverbindung1 102, 106, 107, 190, 302, 331, 343 AuftragLoeschbar 296 AuftragLoeschbar1 344 Auftragsidentifikation 296, 336 Auftragsidentifikation1 302, 343 AuftragsReferenz 335

Ausfuehrungstag1 331, 343 AusfuehrungstageProMonat 111, 310 AusfuehrungstageProWoche 310 Ausgabe formatieren 130 Aussetzung1 344

### В

B2B 355 BACAccount 69, 83, 90, 268, 427 BACAccountData 67 BACBankdata 108 BACBankData 68, 308 BACBanking 44, 45, 330, 368 BACBPDSegmentVorhanden 314 BACContact 61, 65, 193, 427 BACContact-Objekte per Kombinationsfeld 70 BACCustomer 48, 50, 61 BACCustomer anlegen 52 BACCustomer löschen 51 BACCustomers 50 BACDialog 44, 112, 193, 194, 271, 291, 341 BACDialog: Gültigkeit festlegen 113 bacDialogIdle 195 bacDialogReady 195, 212 bacDialogResultCancelled 73 bacDialogResultOK 73 BACDialogResults 73 BACDialog starten 114 BACIbanKonverterResponse 388, 390 BACMessage 44, 118, 271, 333, 341 BACSegment 44, 90, 95, 96, 126, 332 BACSegmentePruefen 152 BACSepaMessage 271, 285, 298, 303, 333, 430, 431 BACSepaOrder 271, 285, 333, 430 BACStatementLine 507 BACSwiftStatementLine 209 BACTransaction 44, 123, 126, 335

BalanceRequest 462, 500, 501 BalanceRequestResponse 502 BankCode 48, 49, 64, 146 BankData 68, 108, 314 Bankinformationen abfragen 476 Bankleitzahl konvertieren 385 BankName 66 Bankparameterdaten 60, 107, 313 BankUserID 66 Bankverbindungen einlesen 136 Bank-Verbindung mit einmaliger Authentifizierung 193 BasisdatenAktualisieren 318 Basislastschrift 353 Basis-Lastschriften durchführen 362 BeginDialog 114, 154, 190, 195, 302, 330 Bericht zur Lastschriftumstellung 405 **BIC 146** aus der Bankleitzahl ermitteln 390 aus der IBAN ermitteln 391 BIC1 302, 331, 343 **BICAusBLZ 390 BICAusIBAN 391** BIC für eine BLZ ermitteln 464 BIC für IBAN ermitteln 473 BICSwiftStatementLine 213 bin 100 Bis1 344 BPD 54, 60, 107, 313 BPDSegment 110, 309, 316 **BPDVersion 66** BusinessTransactionCode 213 BusinessTransactionText 213

# С

CanActivateTANList 74 CanChangeKeys 67 CanUpgradeKeys 67 cboTabelle 394 ChangeConnection 80 ChangeHbciVersion 80 ChangeKeys 75, 79 ChangePin 75, 78 ChangeTANMethod 75, 79 ChangeUserId 80 ChargesAmount 214 ChargesCurrencyCode 214 CheckIban 388, 462, 475 CheckIBAN 391 Chipkarte 20 clsColumnWidth 262 clsTransaction 216 clsTransactions 208, 213, 214 cmdKontostandAktualisieren 187 colDialog 194 Collection 195 CommunicationsAddress 66 CommunicationsAddressSuffix 66 CommunicationsService 66 Contact 66 ContactData 477 ContactData und Konten ermitteln 485 CORE 355 CountryCode 64, 146 CreateContactDirectRequest 462, 477, 486 CreateContactDirectRequestResponse 487 CreateSoapRequest 466 Creditor Identifier 355 ctr 101 cur 101 Currency 147 CurrencyCode 303, 331 CustomerID 66, 147 CustomerReference 214 Customers 48 Customer synchronisieren 53 CustomerSystemID 66 CustomerSystemStatus 66

### D

dat 101DataDesign DDBAC HBCI Banking ApplicationComponents 42, 43, 44, 136

DataDesign DDBAC IBAN Converter 43 DataDesign DDBAC+ PIN/TAN Security Component 43 DataDesign DDBAC S.W.I.F.T. Format Component 43, 209 DataDir 54 DatumBis1 212 Datum der Mandatsreferenz 361 DatumVon1 212 Dauerauftragdetails1 331 Dauerauftrag durchführen 326, 329 Daueraufträge Bestand abrufen 339 Bestand einlesen 335 Daueraufträge löschen 347 DauerauftragSpeichern 344 Dauerauftrag vorbereiten 327 DDBAC.dll 54 DDBAC IBAN Konverter 387 DDBAC.IBAN.Konverter.dll 385 DDBAC.IBAN.Konverter.tlb 386 DDBAC-Komponente 34 DDBAC Segmente.html 155 ddusers.dat 35 DDUSERS.DAT 54 Debuggen 121 Debugging 36 DEG 100 Delete 78 DeleteCustomer 51 Die Deutsche Kreditwirtschaft 356 dig 100

# Ε

Edit 80 eGebucht 208 Einmal-Lastschrift 356 Einrichtung 17 EinzelbuchungErlaubt 422 EinzelbuchungGewuenscht1 431 Einzellastschrift durchführen 366 Einzugsermächtigungsverfahren 385 EndDialog 114, 154, 291, 304 EndToEndReferenz 242 EquivalentAmount 214 EquivalentCurrencyCode 214 Erst-Lastschrift 356 ErstmalsAusfuehrenAm1 331, 343 Execute 122 ExecuteSegment 116, 179, 190, 212, 303, 333, 341 ExecutionDate 283, 303

# F

Fällige Beiträge ermitteln 418 Fälligkeitsdatum 356 Fehlerbehandlung 435 Fehlercodes 124 fehlerhafte Segmente 129 FehlermeldungAusgeben 304 Fehlermeldungen ausgeben 304 Financial Transaction Services 41 FindContact 50 FindCustomer 50 FindOptimalBPDSegment 94, 154, 190, 211, 271, 302, 330, 368 FindOptimalBPDVersion 422 FindSegmentType 110, 314 FinTS 41 FinTS Segmente.html 155 Firmenlastschrift 353 float 100 FNAL 356, 430 Folge-Lastschrift 356 FormatXML 104 Formular zum Ausführen von Terminüberweisungen 281 frmBeispieleBACAccount 91 frmBeispieleBACContact 72, 79, 80 frmBeispieleBACDialog 114 frmBeispieleBACSegment 102 frmEmpfaenger 260

frmIBANConverter 422 frmIBANKonverter 394 frmIBANRechner 393 frmKontaktdetails 478 frmKontoauswaehlen 420 frmKontoAuswaehlen 420 frmKontoauszugNachDatum 248 frmKontostand 173, 174 frmKontostand AlleKonten 182 frmMitglieder 411, 415 frmOnlinebanking 135, 137, 145, 173 frmOptionen 56 frmSegmente 151, 307 frmSegmentinformationen 170 frmSEPADauerauftraegeVerwalten 337 frmSEPADauerauftrag 315, 327 frmSEPAEinzellastschrift 364 frmSEPATerminueberweisungenVerwalten 292 frmSEPAUeberweisung 251, 254, 260, 263, 436 frmSEPAUeberweisungFehlerbehandlung 436 frmSEPAUeberweisungMitVorlagen 275 frmTerminueberweisungen 291 frmUmsaetze 208, 220, 227 frmUmsatzDetails 230 frmWSFunktionen 464 frmWSKonten 478, 510 frmWSUeberweisung 510 FromAccount 271, 303 FRST 356, 430

# G

GeneratelniLetter 75, 79 GetAccountFromIBAN 428 GetAccountList 87, 255, 317 GetAccounts 89, 141 GetAccountsList 88 GetAufsetzpunkt 237 GetBankInfo 453, 462 GetBankInfoResponse 454 GetBankInfoResult 454 GetBanking 45, 57, 190, 302, 375 GetContactFromIBAN 427 GetContactList 71, 136, 255 GetContactListWithFeature 313 GetContactListWithFeatures 338 GetContacts 210, 302 GetDialog 194 GetXML 67, 91, 303, 332 GetXMLAttribut 342 GetXMLElement 296, 303, 342 Gläubiger-ID 361 GläubigerID 355 Gläubiger-Identifikationsnummer 357, 404 Grundlagen der DDBAC 41

### Н

HBCI 41 HBCILog.txt 54 HBCI Protokolldatei aufzeichnen 36 HBCI Segmente.html 155 HBCI+ Segmente.html 155 **HBCIVersion 66** HIBPA 60 HICDBS 338 HICDE 335 HICDES 310, 313 HICSE 290 HIDAES 309 HIDSES 368 HIKAZ 96 HIKAZS 95 HIKOM 60 HIRMG 119 HIRMS 334, 381 HISPAS 294 HITAN 335 HIUPA 61 HIUPD 61, 106 HKCCS 271 HKCDB 308, 340 HKCDE 308, 330, 331 HKCDL 308, 350

### HKCML 122 HKCSB 294 HKCSE 283, 285 HKCSL 302 HKDMB 375, 376 HKDME 429 HKDML 380 HKDSE 368 HKKAZ 95, 96, 150, 154, 207, 232 HKSAL 96, 106, 116 HKVVBVersion 66 Homebanking Computer Interface 41 Homebanking-Kontakt erstellen 17

# I

**IBAN 147** aus BLZ und Kontonummer ermitteln 389 prüfen 391 IBAN1 302, 331, 343 IBANAusBLZUndKontonummer 389 IBANPruefen 391 IBAN prüfen 474 IBANUndBICAktualisieren 292 IBAN und BIC ermitteln 470 id 101 InstallDir 54 InstitutionReference 214 intLastAccount 144 intLastContact 143 IsPersistant 80 IsSepa 88, 147 IsTraceOn 54 ITanSupported 66 **ITANVerfahren** 66

### J

JaehrlichWiederkehrend1 344 jn 101

### Κ

Konfiguration 35 Kontakte durchlaufen 187 Kontaktverwaltung öffnen 46 Konten durchlaufen 188 KontenEinlesen 490 Konten per VBA verwalten 135 Kontoauszug 208, 218, 238 KontoauszugMitAufsetzpunkt 233 Kontoauszugsbericht 242 Kontoinformationen einlesen 480 Kontonummer aus der IBAN ermitteln 392 konvertieren 385 Kontonummer1 102, 107, 190, 343 KontonummerAusIBAN 392 Kontoproduktbezeichnung1 191 Kontostand abrufen 477 Kontostand aktualisieren 187 Kontostand einlesen 498 Kontostand ermitteln 173 KontostandErmittelnAlle 188 KontostandErmittelnKontakt 188, 192 KontostandErmittelnKonto 196 Kontostand für mehrere Konten einlesen 189 Kontoverbindung1 107 Kontowaehrung1 107 Kreditinstitutcode1 107, 190, 343 Kundenreferenz 242

# L

Laenderkennzeichen 1 107, 190, 343 Lastschrift einliefern 356 löschen 378 Lastschriften verwalten 372 Lastschriften einreichen 427 Lastschriften mit SEPA 353 LastschriftSpeichern 377 Lastschrift starten 364

Lastschrifttyp festlegen 382 Lastschrift vormerken 424 Letzten Account speichern 142 Letzten Contact speichern 142 LetzterBankkontakt 255 Letztmalige Lastschrift 356 LetztmalsAusfuehrenAm1 331, 343 LetztmalsAusfuehrenAmAktualisieren 325 IngAuftragID 285, 291 Load 80 LoadXML 468 LockKeys 75, 79 LockPIN 75, 79 LockTANList 75, 78

### Μ

Mandantsreferenz 355 Mandatsreferenz 242, 357, 360, 361 Mandatsreferenznummer anlegen 404 ManualITan 66 ManualUPD 66 Massenweise Konvertierung zu IBAN/BIC 394 MaxAnzahlAuftraege 111, 310 MaxAnzahlInformation 422 MaxAnzahlVerwendungszweckzeilen 111, 310 MaxPinLength 66 MaxVorlaufzeit 111, 310 MaxVorlaufzeitFNAL 368, 421 MaxVorlaufzeitFRST 368, 421 mdlDateifunktionen 58 mdlIBANRechner 389 mdlKontoauszug 207, 208 mdlOBMA SEPADauerauftraege 329 mdlOBMA SEPALastschriften 366 mdlOBMA SEPATerminueberweisungen 282 mdlOBMA SEPAUeberweisungen 269 mdlOnlinebanking 174 mdlOnlinebanking Objekte 194 mdlTools 220 MinAnzahlSignaturen 111, 310

Mindestvorlaufzeit 323 MinPinLength 66 MinVorlaufzeit 111, 310 MinVorlaufzeitFNAL 368, 421 MinVorlaufzeitFRST 368, 421 Mitgliedsbeiträge speichern 411 Mitgliedsdaten speichern 409 MoneyTransferComplete 463 MoneyTransferRequest 462

### Ν

NameEins 147 NameZwei 147 NeedSynchronisation 66 Neuen Kontakt einrichten 47 NewCustomer 52 NewDialog 82 NewDialogUI 81, 114, 152, 193, 210, 271, 302, 330 NewSegment 97, 190, 271, 302, 330 NewWizard 78 NextClosingBalanceAmount 237 NextClosingBalanceCurrencyCode 237 NextClosingBalanceDate 237 num 100 Nur bestimmte Konten anzeigen 201

# 0

Objektmodell 44 Objektmodell der DDBAC 44 Onlinebanking 41 OOFF 356, 430 Optionen von Access aus einstellen 56 Options 54 Orders Add 303 Ordersegment 120 OrderSegment 120 OriginalAmount 214 OriginalCurrencyCode 214

### Ρ

pain.001.002.03 342 Passphrase 23 PersistBPD 54 PersistUPD 54 PIN 114, 252 gültig 441 PIN ändern 31 PIN-Funktionen 31 PIN gesperrt 436 PinOnlyNumeric 66 PinRequirements 66 PIN-Sperre 440 PIN-Sperre aufheben 31 PIN sperren 31 PIN/TAN 20 PIN ungültig 441 Populate 64, 137 Pre-Notification 355 PreviousOpeningBalanceAmount 237 PreviousOpeningBalanceCurrencyCode 237 PreviousOpeningBalanceDate 237 PrimaNoteNumber 214 ProductName 66 ProductVersion 66 Protokollierung 125 Purpose 303, 331 PutXML 91

# Q

qryMitgliedsbeitraege 424, 426 qryUpdateMandatsreferenznummer 404

### R

RCUR 356, 430 RealValidityDate 214 Regasm.exe 386 RemainingSignatures 67 Request 468 RequestTANList 75, 78 **ResponseSegment 335** ResponseSegments 116, 117, 120, 121, 190, 212.287 **ResponseTimeout 54** RetrieveAccountNumberAndBankCode 388 RetrieveBicForBankCode 388, 390, 463 RetrieveBICForBankCode 465 RetrieveBicForBankCodeResponse 469 RetrieveBicForIban 388, 463 RetrieveBicForIbanResponse 473 RetrievelbanAndBic 388, 463, 472 RetrievelbanAndBicResponse 473 Reversal 214 rptKontoauszug 245 rptKontostaende 199, 200, 203 rptKontostaende Auswahl 203, 204 rptUmstellungLastschrifteinzug 405, 407, 408 RunContactsCPL 46 RunNewContactWizard 47

# S

Sammellastschriften einlesen 375 Sammellastschrift mit mehreren Aufträgen 409 Schlüsseldatei 20 Schlüsselpaar 21 Schlüsselphrase 114 SecurityMediaDescription 66 SecurityMediaID 66 SecurityMediaType 66 SecurityProgID 66 Segment 90, 303 SegmenteEinlesen 158 Segment-Informationen anzeigen 168 Segmentparameter einlesen 160 Segments 106, 314 SEPA-Basislastschrift 353 SEPA Business to Business Direct Debit 353 SEPA Core Direct Debit 353 SEPA-Daueraufträge 307

SEPADauerauftragErstellen 332 SEPADescriptor 333 SepaDescriptor1 341 SEPAEinzellastschrift 365, 366 SEPA-Firmenlastschrift 354 SEPA-Lastschriften Datenmodell 383 verwalten 385 SEPA-Lastschriftmandat 356 SEPAPainMessage 333, 341 SepaPainMessages1 341 SEPASammelastschriftenEinlesen 375 SEPASammellastschriftLoeschen 379 SEPASammellastschriftMulti 428 SEPA-Terminüberweisungen 281 SEPA-Überweisung 251 SEPAUeberweisung 269, 270 sfmEmpfaenger 260, 261 sfmIBANKonverter 394 sfmKontostaende 181 sfmKonvertierung 400 sfmMitglieder 414 sfmMitgliedsbeitraege 414 sfmSegmente 151 sfmSegmentparameter 162 sfmSEPADauerauftraegeAbrufen 339 sfmSEPADauerauftraegeVerwalten 336 sfmSEPASammellastschriftenAbrufen 374 sfmSEPATerminueberweisungenVerwalten 293, 300 sfmTerminueberweisungen 291 sfmUmsaetze 222 ShowTANList 78 Sicherheitsfunktion 66 Sicherheitsklasse 310 SignatureID 66 SMS 33 soapclient 455 SOAP-Request 455 SoapUI 458 StartdatumDauerauftrag 323 State 195 StatementLines 213, 507 StatementRequest 463, 505 StatementRequestResponse 507

SubAccountNumber 147 Suchfunktion 164 Summenfeld1 432 SummenfeldErforderlich 422 SupplementaryDetails 214 Synchronize 78, 80 SynchronizeCustomer 53

### Т

Tabelle zum Speichern der Daueraufträge 335 Tabelle zum Speichern der Umsatzdaten 219 TAN 253 TAN-Funktionen 31 TAN-Information 31 TAN-Informationen abfragen 32 TAN-Liste aktivieren 31, 72 TAN-Liste anfordern 31 TAN-Liste sperren 31 **TANProzess** 335 TAN-Verfahren auswählen 35 TARGET2-Geschäftstage 356 tblAnreden 407 tblAuftraege 284, 290, 291, 298 tblBeitragsgruppen 407, 409, 410, 425 tblDauerauftraege 335, 340, 349 tblDaueraufträge 336 tblEmpfaenger 256, 327 tblEmpfaengerBasis 256 tblKontakte 479 tblKonten 483 tblKontostaende 181, 191, 198, 202 tblLastschriften 349, 373, 377, 384 tblMitglieder 396, 404, 405, 407, 409, 414, 416, 425 tblMitgliedsbeitraege 409, 411, 412, 414, 416, 417, 420, 425, 430, 433 tblMitgliedsbeiträge 412 tblOptionen 142, 184, 255 tblOptionenLastschrift 403, 404, 412, 414, 420, 422 tblProtokoll 127, 332, 377 tblSammellastschriften 383

tblSegmente 150, 152, 154 tblSegmentelemente 155, 157 tblUmsaetze 219, 220, 504 tblVorlagen 275 Terminierte SEPA-Sammellastschrift einreichen 429 Terminierte SEPA-Überweisung einreichen 285 TerminierteSEPAUeberweisung 269, 281, 282 TerminierteSEPAUeberweisungenAbrufen 293 Terminüberweisung ändern 305 löschen 299 Terminüberweisung ausführen 282 Terminüberweisungen einlesen 293 Terminüberweisungen abrufen 291 Terminüberweisungen verwalten 291 TerminueberweisungLoeschen 301 TextKeySupplement 214 tim 101 tKontostand 174 ToAccount 303, 331 Transaction 120.304 TransactionNeedsTAN 82 Transactions 117, 120, 190, 212, 271, 287, 333 TransactionType 214 TransaktionAnalysieren 126, 179, 304, 377 Transaktionen ausführen 171 Transaktionen des Kontos ermitteln 149 Turnus1 331, 343 TurnusInMonaten 111, 310 TurnusInWochen 310 txt 100 txtUeberweisungsdatum 281

# U

Überweisung als Vorlage speichern 274 Überweisung anstoßen 266 Überweisung ausführen 269

Überweisungbetrag prüfen 263 Überweisungsdaten 263 Überweisungsformular 251 Überweisungsformular erstellen 254 UmsaetzeGebucht1 212 UmsaetzeNichtGebucht1 212 UmsaetzeVerarbeiten 508 Umsätze einlesen 207, 503 Umsätze im Formular anzeigen 224 Umsatzübersicht per Bericht 197 Umstellung auf SEPA-Lastschriften 403 UnlockPIN 79 UnterformularAktualisieren 184 Unterkontomerkmal1 190, 343 UPD 54,60 **UPDVersion 66** UpgradeKeys 75, 79 UserID 64 Userparameterdaten 60

### V

ValidityDate 214 VBA-Zugriff 42 vdat 101 vElement 102 Verify 271, 332 Version 54, 302, 330 Verwalten der Konten per VBA 148 Verweise-Dialog 42 Verwendungszweck 264 Verwendungszweck auf ungültige Zeichen prüfen 265 vGroup 101 Von1 344 Vorabinformation 355 Vorlage auswählen 279 Vorlage speichern 276

# W

Webservice 453

### Index

Webservice aufrufen 486 WeitereStartdatenDauerauftrag 326 wrt 101

# Χ

XMLAusgabe 130 XMLAusgabeRek 132 XMLHTTP60 468

# Ζ

ZeichenPruefen 266 Zeiteinheit1 331, 343 ZeitraumdatenAktualisieren 321 Zugangsarten 21 Zugriff auf BACContact 64 zuverlässige Variante 192 Zwei-Schritt-TAN-Verfahren 32