Prüfzifferberechnung von Kontonummern C-/Perl-/PHP-Bibliothek

Michael Plugge

13. Mai 2009

Zusammenfassung

Die *konto_check* Bibliothek errechnet anhand von Kontonummer, Bankleitzahl und Prüfziffermethode ob eine angegebene Kontonummer plausibel ist (Prüfzifferverfahren). Es sind alle aktuell definierten 133 Prüfziffermethoden (00 bis D3; Stand März 2009) implementiert. Die Versionen bis 2.7 der Bibliothek waren auf den Test einer Kontonummer beschränkt; ab Version 2.9x/3.0 ist es möglich, zu einer gegebenen Bankleitzahl alle Felder der Bundesbankdatei abzufragen (falls diese in der blz.lut Datei enthalten sind).

Die Bibliothek benutzt für die Informationen zu den einzelnen Banken ein eigenes (komprimiertes) Format. In den Versionen 1.x und 2.x von *konto_check* enthält diese Datei (blz.lut) nur die Bankleitzahlen und zugeordneten Prüfziffermethoden; ab der Version 3.0 können alle Daten aus der Datei der Deutschen Bundesbank. Das Format ist sehr flexibel; es kann eine beliebige Kombination der Felder in der LUT-Datei gespeichert werden. Es ist möglich, nur die Informationen der Hauptstellen aufzunehmen, oder auch die Nebenstellen mit einzubeziehen.

Außerdem können auch eigene Datenblocks in die LUT-Datei geschrieben werden; (z.B. Konfigurationsdaten o.ä.); diese werden natürlich nicht verarbeitet, sondern nur als Ganzes geschrieben bzw. gelesen. Die zugehörigen Funktionen sind momentan nur unter C verfügbar, da dafür noch einige Tests ausstehen.

Diese Datei gibt einen kurzen Überblick über die verfügbaren Funktionen. Sie ist vor allem als Kurzreferenz für die Version 3 gedacht; da die Bibliothek jedoch voll abwärtskompatibel ist, ist sie mit Einschränkungen auch für die alten Versionen zu benutzen.

Anmerkung: Diese Datei bezieht sich vor allem auf die C- und PHP Version. In der Perl-Version gibt es – aufgrund der unterschiedlichen Sprachstruktur – größere Unterschiede. Für diese Version ist sie momentan mehr als Zusatzinfo gedacht; mittelfristig soll allerdings auch die Perl Dokumentation hier integriert werden. Das Perl API ist schon teilweise dokumentiert (via perldoc); die Dokumentation findet sich in der Perl-Distribution.

Diese Beschreibung ist noch sehr unvollständig, und es gibt auch noch manche Wiederholungen; sie wird nach und nach erweitert werden. Aber auch eine nur sehr mäßige Dokumentation ist besser als gar keine. Falls Sie Fehler oder Unvollständigkeiten finden, wäre es schön, falls Sie sie mir eine kurze Nachricht schicken: m.plugge [at] hs-mannheim.de

Inhaltsverzeichnis

1 Paketliste **1**

2 Überblick, Anmerkungen zur Version **1**
2.1 Einführung . . . 1
2.2 Aktuelle Version . . . 1
2.3 Installation . . . 2
2.4 Bekannte Fehler . . . 2
2.5 Technische Informationen, Geschwindigkeit . . . 2
2.6 Unterschiede zwischen *konto_check* 2.x und 3.0 . . . 3

3 Die Datendatei blz.lut **3**
3.1 Format der LUT-Datei . . . 3
3.2 Aktualisierung der Bankdaten . . . 5

4 Für Ungeduldige: einige Minimalbeispiele **5**
4.1 Mini-Programm in C . . . 5
4.2 Mini-Programm in Perl . . . 6
4.3 Mini-Programm in PHP (altes Interface) . . . 6
4.4 Mini-Programm in PHP (neues Interface) . . . 7

5 Wichtige Funktionen der C Bibliothek (nicht komplett) **9**
5.1 Einführung, Kompatiblität zur Vorversion . . . 9
5.2 Initialisierung der Bibliothek . . . 9
5.3 Aufrufparameter: . . . 10
5.4 Konvertierung der numerischen Rückgabewerte in Klartext . . . 10
5.5 Test einer Bankverbindung . . . 11
5.6 Test von IBAN und strukturiertem Verwendungszweck . . . 11
5.7 Abfrage von Bankdaten (aus der BLZ) . . . 11
5.8 Sonstige Funktionen . . . 12

6 Die PHP-Version von *konto_check* **12**
6.1 Einführung . . . 12
6.2 Installation der PHP-Version . . . 13

7 Das alte PHP API **13**
7.1 Initialisierung, Infos zur LUT-Datei, Speicherfreigabe . . . 13
7.2 Konvertierung der numerischen Rückgabewerte in Klartext . . . 14
7.3 Testfunktionen: BLZ/Prüfziffer und Konto . . . 14
7.4 Testfunktionen: IBAN, Strukturierter Verwendungszweck . . . 15

8 Das neue PHP API **16**
8.1 Einführung . . . 16
8.2 Initialisierung, Infos zur LUT-Datei etc., Speicherfreigabe . . . 17
8.3 Konvertierung der numerischen Rückgabewerte in Klartext . . . 19
8.4 Testfunktionen: BLZ/Prüfziffer und Konto . . . 19
8.5 Testfunktionen: IBAN, Strukturierter Verwendungszweck . . . 20
8.6 Abfrage von Bankdaten (Felder der Bundesbank-Datei) . . . 21

9 Die Perl-Version von *konto_check* **22**
9.1 Einführung . . . 22
9.2 Installation der Perl-Version . . . 22

10 Die Windows-Version von *konto_check* (DLL, VBA) **23**

11 Anhang: Diverse Tabellen **23**
11.1 Felder der Bundesbankdatei bzw. der LUT-Datei 23
11.2 Namen der Datenblocks 26
11.3 Datenblocks für verschiedene (skalare) Werte von `required` 27
11.4 Rückgabewerte der Bibliothek 27
11.5 Klartextbeschreibung der Rückgabewerte 27
11.6 Makronamen bzw. Kurznamen der Rückgabewerte (alphabetisch sortiert) 30

12 VERSIONEN **31**

13 Copyright **32**

Allgemeiner Index **33**

Index der C-Funktionen **34**

Index der PHP-Funktionen (alt) **35**

Index der PHP-Funktionen (neu) **36**

1 Paketliste

00liesmich.pdf:	diese Datei
latex/∗	Quellcodes für diese Datei
0_history.txt:	Versionsgeschichte
blz.lut:	Lookup-Table für Bankleitzahlen (alte Version)
blz.lut2:	Lookup-Table für Bankleitzahlen (neue Version)
konto_check.c:	C-Datei (komplettes Prüfziffermodul)
konto_check.h:	Header-Datei mit public interface
lgpl.txt:	GNU Lesser General Public Lizenz
lgpl-ger.html:	deutsche Übersetzung der LGPL
main.c:	Beispielsprogramm
konto_check_mini.c:	weiteres Beispielsprogramm
mini.c:	noch ein Beispielsprogramm
mini.pl:	dasselbe, in Perl
mini.old.php:	dasselbe, in PHP (altes API)
mini.php:	dasselbe, in PHP (neues API)
makefile.unix:	Makefile für Unix
php.zip:	PHP-Version von konto_check (kurze Anleitung in 00build.txt)
php.old.zip:	PHP-Version von konto_check, altes API
testkonten.txt:	diverse Testkonten (mit Prüfziffern)
testkonten.blz:	diverse Testkonten (mit BLZ)

2 Überblick, Anmerkungen zur Version

2.1 Einführung

Die *konto_check*-Bibliothek errechnet anhand von Kontonummer und Bankleitzahl (bzw. Prüfziffermethode), ob eine angegebene Kontonummer plausibel ist. Die Versionen 1.x und 2.x der Bibliothek waren auf den Test einer Kontonummer beschränkt; ab Version 2.9x/3.0 ist es möglich, zu einer gegebenen Bankleitzahl alle Felder der Bundesbankdatei abzufragen, falls diese in der blz.lut Datei enthalten sind.

Die Bibliothek benutzt für die Informationen zu den einzelnen Banken ein eigenes (komprimiertes) Format. In den Versionen bis 2.7 von *konto_check* enthält diese Datei (blz.lut) nur die Bankleitzahlen und zugeordneten Prüfziffermethoden; ab der Version 3.0 können alle Daten aus der Datei der Deutschen Bundesbank, sowie auch eigene Dantenblocks in der blz.lut Datei gespeichert werden. Das Format ist sehr flexibel; es kann eine beliebige Kombination der Felder in der LUT-Datei gespeichert werden. Es ist möglich, nur die Informationen der Hauptstellen aufzunehmen, oder auch die Nebenstellen mit einzubeziehen. Eigene Datenblocks werden natürlich nicht verarbeitet, sondern nur als Ganzes geschrieben bzw. gelesen.

Diese Datei gibt einen kurzen Überblick über die verfügbaren Funktionen. Sie ist vor allem als Kurzreferenz für die Version 3 gedacht; da die Bibliothek jedoch voll abwärtskompatibel ist, ist sie auch teilweise für die alten Versionen brauchbar.

2.2 Aktuelle Version

Die aktuelle Version (2.9x) wurde intensiv mit der vorhergehenden Version (2.6/2.7) verglichen (mit ca. 150 Millionen Testkonten :-)); dabei wurden einige Fehler in der alten Bibliothek gefunden, aber ansonsten keine Unterschiede festgestellt. Auch die Routinen für das neue LUT-Dateiformat scheinen keine Probleme zu bereiten; daher wird eine der nächsten Versionen dann die 3.0 werden. Außerdem wurden die neu definierten Prüfmethoden implementiert; es werden alle 133 definierten Prüfmethoden unterstützt (00 bis D3; Stand Februar 2009).

Der Windows-Port macht unter PHP und Access noch etwas Probleme; auf der Plattform habe ich allerdings schon länger keine Tests mehr gemacht, das müßte noch getan werden.

Das Beispielsprogramm kann auch als Filterapplikation verwendet werden. Mit dem Befehl `./konto_check -h` wird eine kleine Online-Hilfe ausgegeben. Näheres zu den Optionen ist in den Sourcen (vor allem main.c) zu finden. Das Eingabe- und Ausgabeformat wurde etwas geändert, so daß eine Eingabezeile immer an die Ausgabedatei weitergereicht wird, und (falls eine Bankleitzahl und Kontonummer gefunden wurde) noch das Testergebnis angehängt wird. Dieses kann vom Eingangstext durch ein frei wählbares Trennzeichen (Option -s) getrennt werden. Bei Benutzung von Prüfziffermethoden ist bei der Berechnung der Methoden 52, 53, B6 und C0 auch noch die Bankleitzahl von Bedeutung.

2.3 Installation

Die *konto_check* Bibliothek besteht aus einer einzigen großen C-Datei, die als Objektcode zu einem anderen Programm gelinkt werden kann. Prototypen und das API können der Datei konto_check.h entnommen werden; dort sind für die meisten Funktionen kurze Beschreibungen enthalten. Eine kurze Beschreibung einiger wichtiger Funktionen erfolgt auch in dieser Datei. Eine eigentliche Installation ist nicht erforderlich, sie kann jedoch bei Bedarf (z.B. als shared library) erfolgen.

Eine Aufteilung der Datei in mehrere Module ist weder sinnvoll noch einfach möglich. Der größte Teil des Codes besteht aus den Prüfzifferroutinen, die in einem großen switch Statement zusammengefaßt sind. Daneben gibt es noch eine Reihe von statischen Variablen, in denen die Werte der LUT-Datei gespeichert sind. Bei einer Aufteilung der Datei in mehrere Module müßten diese Variablen entweder lokal oder global deklariert werden, was in beiden Fällen gravierende Nachteile mit sich bringen würde.

2.4 Bekannte Fehler

Momentan sind in den Prüfziffermethoden keine Fehler bekannt (was nicht heißt, daß keine da sind!). Auch die LUT2-Routinen scheinen stabil zu sein; die Standardoperationen sind relativ gut getestet, für einige (eher selten benutzte) Sonderfunktionen stehen noch einige Tests aus. Falls Sie einen Fehler finden, würde ich mich sehr freuen, davon zu hören ;-))). Bugs und Anregungen bitte an m.plugge@hs-mannheim.de

2.5 Technische Informationen, Geschwindigkeit

Die Bankleitzahl ist immer 8-stellig. Die Länge von Kontonummern ist variabel; diese werden auf 10 Stellen erweitert, indem führende Nullen ergänzt werden.

Die Datei testkonten.txt ist eine Testdatei mit knapp 600 Kontonummern und Bankleitzahlen Einige sind reale Kontonummern, andere sind Testkonten, die von der Deutschen Bundesbank zur Verfügung gestellt werden. Im ersten Feld ist die jeweilige Bankleitzahl oder Prüfziffer zu finden, im zweiten Feld die Kontonummer.

Die Bibliothek wurde mit allen Testkontonummern der Dokumentation der Prüfzifferberechnungmethoden der Deutschen Bundesbank erfolgreich getestet. Außerdem wurden etliche Testkontonummern generiert und die Ergebnisse mit mehreren anderen Programmen verglichen; dabei konnten (auf beiden Seiten) viele Fehler aufgespürt werden.

Die Bibliothek wurde für die Versin 2.0 stark auf Geschwindigkeit hin optimiert. Bei den Tests wurde (unter Linux) eine Datei mit 5.000.000 Kontonummern in 1,1s geprüft (auf einem 3,4 GHz Pentium 4; Kompilierung mit eingeschalteter Optimierung, ohne Debuginfo; Aufruf `./konto_check -f testkto.blz testkto.out`. Ohne Optimierung, mit -f werden etwa 1,7s benötigt, ohne Optimierung und ohne -f etwa 2,6s). Die Version 2.95 ist wieder etwas langsamer und benötigt (auf demselben Rechner) 1,3s für diese Aufgabe.

2.6 Unterschiede zwischen *konto_check* 2.x und 3.0

Außer dem neuen Dateiformat wurde das Aufrufinterface der Bibliothek noch einmal komplett überarbeitet:

- in der alten Version gab es eine Funktion zur Prüfung, die sowohl Prüfziffermethoden (zwei- bzw. dreistellig) als auch Bankleitzahlen (achtstellig) verarbeitete. Außerdem diente die Funktion zusätzlich noch zur Initialisierung; falls die Bibliothek noch nicht initialisiert war, wurde sie beim Aufruf der Testroutine initialisiert.

  Im neuen Interface gibt es dagegen für Prüfziffermethoden und Bankleitzahlen jeweils eigene Funktionen. Die Initialisierung ist von den Testroutinen getrennt; die Bibliothek muß jetzt *vor* der Prüfung initialisiert werden (falls nicht das alte Interface benutzt wird).

  Eine Initialisierung kann sowohl als Neuinitialisierung (alle evl. noch vorhandenen Blocks werden gelöscht) als auch inkrementell erfolgen (falls später noch zusätzliche Blocks benötigt werden, können diese noch zusätzlich aus der LUT-Datei gelesen werden, ohne daß die bislang initialisierten Daten verworfen werden müßten). Die inkrementelle Initialisierung ist aus Konsitenzgründen nur von derselben LUT-Datei möglich; zur Identifikation enthält daher jede Datei eine eigene (zufällige) Datei-ID.

- die alte Version war nicht von vornherein threadfest, da für einige Rückgaben globale Variablen benutzt werden; sie ließ sich zwar (mit einigen schmutzigen Tricks) auch threadfest machen, aber dies war nur durch Einführung einer zusätzlichen (Struktur-) Variablen möglich.

  In der neuen Version wurden sämtliche globalen Variablen, auf die aus den Prüfzifferroutinen schreibend zugegriffen wird, entfernt und durch lokale Variablen ersetzt; dadurch sind alle Testroutinen threadfest.

  Die einzige Ausnahme sind die Funktionen zur Generierung einer LUT-Datei, die (aufgrund der Quicksort-Funktion der libc) globale Variablen zwingend benötigt. Dies sind allerdings auch nicht kritisch, da die globalen Variablen der Sortierroutinen nicht mit Variablen der Testroutinen kollidieren und ein mehrfacher Aufruf der Funktionen zur Generierung einer LUT-Datei nicht sinnvoll ist und normalerweise nicht vorkommen dürfte.

- in der alten Version ist es nicht einfach möglich, die Bibliothek neu zu initialisieren; falls dies notwendig ist, muß u.U. der entsprechende Prozess beendet und neu gestartet werden. Die neue Version kann problemlos sowohl komplett als auch inkrementell (nur einzelne Felder) neu initialisiert werden.

- das alte LUT-Format erlaubt nur einen Datensatz; im neuen Format können zwei Datensätze mit ihrem jeweiligen Gültigkeitszeitraum gespeichert werden (das war eine Anregung von byteripper[at]users.sourceforge.net). Die Auswahl des Datensatzes bei der Initialisierung kann automatisch (aufgrund des Systemdatums) oder manuell erfolgen.

- die neue Version enthält noch einige Funktionen für IBAN und IPI (Test einer IBAN, Bestimmung des BIC aus einer IBAN (nur für deutsche Bankleitzahlen), Validierung eines strukturierten Verwendungszwecks, Generieren eines strukturierten Verwendungszwecks).

3 Die Datendatei blz.lut

3.1 Format der LUT-Datei

Die Datei blz.lut enthält eine komprimierte Version von Daten der Bankleitzahlendatei der Deutschen Bundesbank. Es sind drei Formate definiert:

- Das Format 1.0 war das erste Format; es ist mittlerweile obsolet, da die anderen Formate benutzerfreundlicher sind.

  Die Datei besteht aus einer Identifikationszeile im Klartext (`BLZ Lookup Table/Format 1.0`); danach kommen direkt die Bankdaten. Es sind nur BLZ und Prüfziffermethode gespeichert. Für die Bankleitzahlen wird üblicherweise die Differenz zur vorigen BLZ gespeichert (meist mit 1 oder 2 Byte), danach kommt ein Byte für die Prüfziffermethode.

- Das Format 1.1 ist eine kleine Erweiterung von 1.0; die Bankdaten sind mit der Version 1.0 identisch; es wird jedoch eine zusätzliche Infozeile geschrieben, wann und aus welcher Datei die LUT-Datei generiert wurde (ebenfalls Klartext). Außerdem kann noch eine benutzerdefinierte Infozeile hinzugefügt werden (z.B. mit dem Gültigkeitszeitraum). Diese Zeile kann von der *konto_check*-Bibliothek ausgelesen werden; da das Format jedoch vollkommen frei ist, kann sie nicht ausgewertet werden.

- Das Format 2.0 unterscheidet sich von den Vorgängerversionen wesentlich:

  - Der Klartextteil des Headers enthält die Daten aus Version 1.1, sowie noch einige zusätzliche Infos (Anzahl Banken, Anzahl Hauptstellen, Angabe, ob nur die Hauptstellen oder auch die Filialen in der Datei enthalten sind).
  - Jede Datei erhält eine eindeutige (zufällige) Datei-ID; diese dient dazu, beim Nachladen von Blocks zu verifizieren, daß nicht von einer anderen Datei Blocks nachgeladen werden, da dies normalerweise zu Inkonsistenzen der Daten führen würde.
  - Der Datenbereich enthält zwei Teile: einen Verzeichnisbereich, dessen Größe bei der Erzeugung der Datei festgelegt wird und nicht mehr verändert werden kann, sowie den eigentlichen Datenbereich mit den Blocks.
  - Im Verzeichnis wird für jeden Block der Blocktyp, die Startadresse innerhalb der Datei, Blockgröße sowie eine Prüfsumme gespeichert. Auf diese Weise können die Blocks unabhängig von einander gelesen werden. Die (internen) Funktionen sehen im Verzeichnisbereich nach, ob und wo ein gewünschter Block innerhalb der Datei zu finden ist, und wie groß er ist, und laden dann genau den einen Block in den Speicher. Die einzelnen Blocks sind mit der *zlib* komprimiert.
  - Innerhalb einer LUT-Datei können zwei verschiedene Bankleitzahlendateien (mit unterschiedlichem Gültigkeitszeitraum) enthalten sein. Die Blocktypen von 1 bis 100 gehören zum ersten Datensatz, die Blocktypen von 101 bis 200 zum zweiten. Jeder Satz enthält einen Info-Block (numerischer Typ 15 bzw. 115), in dem der Dateivorspann für den Block sowie der Gültigkeitszeitraum angegeben ist. Die Initialisierung der *konto_check*-Bibliothek kann dann in Abhängigkeit vom aktuellen Datum erfolgen (dabei wird der Gültigkeitszeitraum der beiden Datensätze überprüft und derjenige Satz zur Initialisierung verwendet, der aktuell gültig ist), oder man kann von einem bestimmten Datensatz initialisieren. In der Funktion lut_init gibt es einen Parameter `set` mit dem Wert 0, 1 oder 2; mit diesem kann man einen Block zur Initialisierung auswählen (1 oder 2) oder die Auswahl abhängig vom Gültigkeitsdatum treffen lassen (0).
  - Für die Blocktypen sind Werte zwischen 1 und 65535 möglich; die Werte zwischen 1 und 1000 sind dabei für die *konto_check*-Bibliothek reserviert, während andere Werte auch von anderen Programmen verwendet werden können (z.B. um bestimmte Einstellungen eines Programms o.ä. zu speichern). Es gibt eine Reihe High-Level-Funktionen, die den Zugriff auf Datenblocks (Lesen und Schreiben) mit wenig Aufwand erlauben. Momentan sind sie nur in der C-Bibliothek verfügbar, da sie noch einige Tests benötigen; später werden sie auch in den andern APIs zugänglich gemacht werden.

3.2 Aktualisierung der Bankdaten

Die LUT-Datei kann mit der Funktion generate_lut() bzw. generate_lut2() aus der Datei der Deutschen Bundesbank generiert werden. Die aktuelle Bankleitzahltabelle wird veröffentlicht von der Deutschen Bundesbank:

http://www.bundesbank.de/zahlungsverkehr/zahlungsverkehr_bankleitzahlen_download.php

Dabei ist die Datei blzJJJJMMTTpc.zip herunterzuladen und auszupacken. Die Aktualisierung erfolgt alle drei Monate. Die aktuellen Prüfziffermethoden werden ebenfalls von der Deutschen Bundesbank veröffentlicht. Dort finden sich auch weitere allgemeine Informationen und Links.

http://www.bundesbank.de/zahlungsverkehr/zahlungsverkehr_pruefziffernberechnung.php

4 Für Ungeduldige: einige Minimalbeispiele

Für diejenigen, die lieber programmieren als Handbücher lesen, hier einige kleine Codebeispiele, die die Verwendung der Bibliothek zeigen. Das C- und Perlprogramm liest aus einer Datei Bankleitzahl/Konto-Kombinationen und schreibt die Prüfergebnisse in eine zweite Datei; die Dateien werden auf der Kommandozeile angegeben. Im PHP-Programm werden die Eingaben in einem Textfeld gemacht und die Ergebnisse nach dem Textfeld wieder ausgegeben. Bankleitzahl und Konto sind in allen Programmen jeweils durch ein oder mehrere Blanks zu trennen.

Die Programme wurden mit `tiny_c2l` nach LaTeX konvertiert und teilweise mit Hyperlinks zu einigen Funktionen versehen. Das `tiny_c2l` ist eigentlich nicht für Perl oder PHP gedacht, funktioniert aber (mit kleinen Abstrichen) trotzdem.

4.1 Mini-Programm in C

```
#include <stdio.h>
#include "konto_check.h"

int main(int argc,char **argv)
{
  char blz[16],kto[16],buffer[256],*name,*ort,*retval_txt;
  int retval,r1,plz;
  FILE *in,*out;

  if(argc<2)return 1;
  if(!(in=fopen(argv[1],"r")))return 2;
  if(!(out=fopen(argv[2],"w")))return 3;

  if((retval=lut_init("blz.lut2",5,0))!=OK){      /* Bibliothek initialisieren */
      /* Rückgabewert als Klartext ausgeben */
    fprintf(stderr,"lut_init: %s\n",kto_check_retval2txt(retval));
    return 4;
  }

  while(!feof(in)){
    fgets(buffer,256,in);  /* Datei zeilenweise einlesen */
    if(sscanf(buffer,"%s %s",blz,kto)<2 || feof(in))continue;
    retval=kto_check_blz(blz,kto);              /* Bankverbindung testen */
    retval_txt=kto_check_retval2txt(retval);    /* Rückgabewert als Klartext ausgeben */
    name=lut_name(blz,0,NULL);                  /* Banknamen holen */
    plz=lut_plz(blz,0,NULL);                    /* PLZ holen */
    ort=lut_ort(blz,0,NULL);                    /* Ort holen */
```

```
    if(retval>=OK)
        fprintf(out,"%s %s: %s; %s, %d %s\n",blz,kto,retval_txt,name,plz,ort);
    else
        fprintf(out,"%s %s: %s\n",blz,kto,retval_txt);
  }
  lut_cleanup(); /* Speicher wieder freigeben */
}
```

4.2 Mini-Programm in Perl

```
use Business::KontoCheck qw(kto_check_init kto_check_blz lut_name
     lut_plz lut_ort lut_cleanup %kto_retval);

($ret=kto_check_init("blz.lut2"))>0
     or die "Fehler bei der Initialisierung: $kto_retval{$ret}\n";

open(IN,@ARGV[0]) or die "Kann ".@ARGV[0]." nicht lesen: $!\n";
open(OUT,"> @ARGV[1]") or die "Kann @ARGV[1] nicht schreiben: $!\n";

while(<IN>){
  chomp;
# eine Zeile aufdröseln und den Variablen zuweisen
  ($valid,$blz,$kto)=/(([0-9]+)[ \t]+([0-9]+))?/;
  if($valid){
    $retval=kto_check_blz($blz,$kto);
    if($retval>0){    # OK -> Banknamen und Adresse ausgeben
      $name="; ".lut_name($blz)."; ".lut_plz($blz)." ".lut_ort($blz);
    }
    else{ # Fehler, leer lassen
      $name="";
    }
    print OUT "$blz $kto: $kto_retval{$retval}$name\n";
  }
}
lut_cleanup(); # Speicher freigeben
```

4.3 Mini-Programm in PHP (altes Interface)

```
<?
function set_var($var)
{
  global $$var;

  if(isset($_REQUEST[$var]))
    $$var=$_REQUEST[$var];
  else
    $$var="";
}

set_var("testkonten");
set_var("testen");

?>
```

```
<html><head><title>Konto_check Testseite (altes API)</title></head>
<body>
<h1>Testseite für konto_check (altes API)</h1>
Bitte in das folgende Textfeld die zu testenden Bankverbindungen
(BLZ/Kto, durch Blanks getrennt) eingeben.

<p><form>
<textarea cols="20" rows="15" name="testkonten">
<?=$testkonten?>
</textarea>
<p><input type="submit" name="testen" value="Konten testen">
</form>

<?
if(!empty($testen) && !empty($testkonten)){
   if(($retval=lut_init())<=0)      /* Bibliothek initialisieren (mit Defaultwerten) */
      die("Fehler bei der Initialisierung: ".kto_check_retval2html($retval));
   echo "<h2>Testergebnisse:</h2>\n";
   $zeilen=explode("\n",$testkonten);          /* Textfeld in Zeilen aufsplitten */
   foreach($zeilen as $i=>$v){
      $v=trim($v);
      if(!empty($v)){
         list($blz,$kto)=split("[ \t]+",$v);      /* Zeile aufdröseln */
         if(!empty($blz) && !empty($kto)){
            $retval=kto_check_blz($blz,$kto);    /* Bankverbindung testen */
            $test=kto_check_retval2html($retval);
               /* die Rückgabewerte der folgenden Funktionen sind Arrays */
            $name=lut_name($blz); /* Bankname */
            $plz=lut_plz($blz);      /* PLZ der Bank */
            $ort=lut_ort($blz);      /* Ort der Bank */
            $ret=$name["ret"];       /* Rückgabestatus von lut_name holen */
            if($ret>0)               /* OK -> Banknamen und Adresse ausgeben */
               $bankname="; ${name['wert']}, ${plz['wert']} ${ort['wert']}";
            else
               $bankname="";         /* Fehler in BLZ; Banknamen leer lassen */
            printf("$blz $kto: $test$bankname<br>\n");
         }
      }
   }
   lut_cleanup(); /* Speicher freigeben */
}
?>
</body></html>
```

4.4 Mini-Programm in PHP (neues Interface)

```
<?
function set_var($var)
{
   global $$var;

   if(isset($_REQUEST[$var]))
      $$var=$_REQUEST[$var];
   else
```

```
        $$var="";
}

    /* Übergabevariablen holen */
set_var("testkonten");
set_var("testen");

?>
<html><head><title>Konto_check Testseite (neues API)</title></head>
<body>
<h1>Testseite für konto_check (neues API)</h1>
Bitte in das folgende Textfeld die zu testenden Bankverbindungen
(BLZ/Kto, durch Blanks getrennt) eingeben.

<p><form>
<textarea cols="20" rows="15" name="testkonten">
<?=$testkonten?>
</textarea>
<p><input type="submit" name="testen" value="Konten testen">
</form>

<?
if(!empty($testen) && !empty($testkonten)){
    if(($retval=lut_init())<OK)      /* Bibliothek initialisieren (mit Defaultwerten) */
        die("Fehler bei der Initialisierung: ".kc_html($retval));
    echo "<h2>Testergebnisse:</h2>\n";
    $zeilen=explode("\n",$testkonten);    /* Textfeld in Zeilen aufteilen */
    foreach($zeilen as $i=>$v){
        $v=trim($v);
        if(!empty($v)){
            list($blz,$kto)=split("[ \t]+",$v);        /* Zeile aufdröseln */
            if(!empty($blz) && !empty($kto)){          /* BLZ und Konto sind notwendig */
                $retval=kto_check_blz($blz,$kto);      /* Bankverbindung testen */
                $rv_txt=kc_html($retval);              /* Rückgabewert als Klartext */
                $name=lut_name($blz,$ret);             /* Banknamen holen */
                if($ret==OK){                          /* Bank gefunden, BLZ ok */
                    $plz=lut_plz($blz);                /* Postleitzahl der Bank */
                    $ort=lut_ort($blz);                /* Ort der Bank */
                    $info="; $name, $plz $ort";        /* Bankinfo zusammenstellen */
                }
                else
                    $info="";     /* Fehler in BLZ; Banknamen leer lassen */
                printf("$blz $kto: $rv_txt$info<br>\n");
            }
        }
    }
    lut_free();  /* Speicher freigeben */
}
?>
</body></html>
```

5 Wichtige Funktionen der C Bibliothek (nicht komplett)

5.1 Einführung, Kompatiblität zur Vorversion

Zum Test eines Kontos stehen sowohl die Funktionen der alten Version (bis 2.7) sowie das der 2.9x/3.0 Version zur Verfügung.

Das alte API ist relativ klein; es enthält einige Funktionen zum Test eines Kontos sowie noch einige Verwaltungsfunktionen (Speicherfreigabe, Generieren einer LUT-Datei, Infozeile einer LUT-Datei holen). Diese Funktionen sind in der neuen Version nicht eigenständig programmiert, sondern rufen intern die neuen Funktionen auf. Sie sind nur noch aus Kompatibilitätsgründen in der Bibliothek enthalten; es ist jedoch empfehlenswert, sie durch die neuen Funktionen zu ersetzen, da bei diesen der Overhead geringer ist.

Bei der alten Version gab es noch eine Reihe Funktionen, bei denen die Threadfestigkeit durch eine zusätzliche Variable (Struktur, in der die ansonsten globalen Variablen gehalten wurden) erreicht wird; diese Funktionen sind mit einem _t im Dateinamen gekennzeichnet und enthalten einen zusätzlichen Parameter vom Typ `KTO_CHK_CTX` (z.B. `kto_check_t`() oder `kto_check_str_t`()). Diese Funktionen sind komplett obsolet, da die Bibliothek jetzt threadfest ist. Sie sind noch in der Bibliothek enthalten, haben aber die gleiche Funktionalität wie die entsprechenden Routinen ohne das _t (der zusätzliche Parameter wird einfach ignoriert). In der folgenden Übersicht sind sie nicht mehr aufgeführt; in der Headerdatei konto_check.h sind sie natürlich noch zu finden.

Ein Punkt, an dem die Kompatibilität allerdings gebrochen werden mußte, sind globale Variablen; diese werden im neuen API nicht mehr unterstützt, da sie nicht threadfest realisiert werden können. Die entsprechenden Variablen existieren noch (um evl. Linkerfehler zu vermeiden); sie enthalten jedoch nur eine Warnmeldung, daß die entsprechende Variable nicht mehr unterstützt wird.

Das neue API enthält enthält wesentlich mehr Funktionen, da zu einer Bankleitzahl alle Felder der BLZ-Datei der Deutschen Bundesbank abgefragt werden können. Hinzu kommen noch Funktionen zum Tests von IBAN und Strukturiertem Verwendungszweck.

Im neuen API sind Initialisierung der Bibliothek und Test getrennt; außerdem ist es möglich, inkrementell zu initialisieren (d.h., zusätzliche Felder einer LUT-Datei nachzuladen). In der neuen LUT-Datei können zwei verschiedene Datensätze mit unterschiedlichem Gültigkeitsdatum enthalten sein. Die Auswahl des Datensatzes kann automatisch (nach Datum) oder manuell erfolgen.

Außerdem ist es möglich, zu testen, ob der Datensatz im Speicher aktuell noch gültig ist. Diese Funktion ist momentan noch nicht wichtig; für Perl und PHP sind allerdings persistente Datenstrukturen geplant, bei denen die LUT-Datei während der Initialisierung *des Servers* geladen wird. In dem Fall wäre die Funktion dann in einem Request aufzurufen, um u.U. (falls die Daten nicht mehr gültig sind) eine neue Initialisierung durchzuführen

5.2 Initialisierung der Bibliothek

Diese Funktionen allokieren Speicher für interne Arrays, lesen die LUT-Datei und tragen die Werte in die Arrays ein. Eine dieser Funktionen muß _vor_ Benutzung der Bibliothek aufgerufen werden. Sie unterscheiden sich nur in den Aufrufparametern; intern rufen die Funktionen lut_init() und kto_check_init_p() die Funktion kto_check_init() auf. (Beispiel)

- `int lut_init(char *lut_name,int required,int set)`

  Diese Funktion dient zur einfachen Initialisierung der Bibliothek.

- `int kto_check_init_p(char *lut_name,int required,int set,int incremental)`

  Diese Funktion hat Ähnlichkeit mit der Funktion lut_init(); der zusätzliche Parameter `incremental` dient zur inkrementellen Initialisierung der Bibliothek. Sie wurde eingeführt, da für Perl keine Arrays als Parameter übergeben werden können, daher auch das `_p` im Funktionsnamen.

- `int kto_check_init(char *lut_name,int *required,int **status,int set,int incremental)`

  Dies ist die allgemeine Funktion zur Initialisierung der Bibliothek. Der Parameter `required` ist hier ein Array; außerdem gibt es noch die Variable `status` in dem der Status jedes einzelnen Blocks zurückgegeben wird.

5.3 Aufrufparameter:

char *lut_name (bei allen Funktionen):

Dateiname der Datei mit den komprimierten Bankdaten (LUT-Datei).

int required (`lut_init()`, `kto_check_init_p()`):

Dieser Parameter (Integerzahl) gibt die zu ladenden Datenblocks an; er kann die Werte 0...9 annehmen. Die Zuordnung zu den zu ladenden Blocks ist im Anhang aufgeführt.

int *required (bei `kto_check_init()`):

Dieser Parameter (als Integer-Array) gibt die zu ladenden Datenblocks an (wie oben); man ist in der Auswahl der einzelnen Blocks jedoch komplett frei. Die Liste muß durch ein Element mit 0 abgeschlossen werden. Die definierten Blocktypen sind im Anhang definiert.

int set (bei allen Funktionen):

Dieser Parametergibt den zu ladenden Datensatz an (0, 1 oder 2). Bei 0 wird der aktuell gültige Datensatz geladen (bzw. falls keiner aktuell ist oder die Datei kein Gültigkeitsdatum enthält, der erste); ansonsten der angegebene Satz. Nähere Angaben zum Gültigkeitsdatum finden sich bei der Funktion generate_lut2 bzw. in der Beschreibung der LUT-Datei.

int **status (bei `kto_check_init()`):

Diese Variable wird auf ein Integer-Array gesetzt, in dem der Status jedes einzelnen Blocks zurückgegeben wird. Damit kann man beim Rückgabewert von `LUT2_PARTIAL_OK` nachsehen, welche Blocks geladen wurden, und welche Fehler bei den fehlenden Blocks auftraten.

int incremental (bei `kto_check_init_p()`, `kto_check_init()`):

Falls dieser Parameter 1 ist, wird versucht, die Bibliothek inkrementell zu initialisieren (d.h., nur die fehlenden Blocks nachzuladen). Falls eine inkrementelle Initialisierung von einer anderen Datei als der ursprünglichen versucht wird, werden keine zusätzlichen Blocks geladen, um Inkonsistenzen zu vermeiden; stattdessen wird die Fehlermeldung `INCREMENTAL_INIT_FROM_DIFFERENT` zurückgegeben.

5.4 Konvertierung der numerischen Rückgabewerte in Klartext

Die folgenden Funktionen konvertieren die numerischen Rückgabewerte in Klartext in verschiedene Formate:

- `char *kto_check_retval2txt_short(int retval)` Diese Funktion wandelt den Rückgabewert retval in einen String um, der dem Makronamen des Rückgabeparameters entspricht.
- `char *kto_check_retval2txt(int retval)` Diese Funktion wandelt den Rückgabewert retval in einen String um. Die Kodierung der Umlaute erfolgt mit ISO-8859-1.
- `char *kto_check_retval2html(int retval)` Diese Funktion wandelt den Rückgabewert retval in einen String um. Die Kodierung der Umlaute erfolgt mit HTML Entities. (Beispiel)
- `char *kto_check_retval2dos(int retval)` Diese Funktion wandelt den Rückgabewert retval in einen String um. Die Kodierung der Umlaute erfolgt mit cp850 (DOS).
- `char *kto_check_retval2utf8(int retval)` Diese Funktion wandelt den Rückgabewert retval in einen String um. Die Kodierung der Umlaute erfolgt mit UTF-8.

5.5 Test einer Bankverbindung

Die folgenden Funktionen sind für das C API noch nicht beschrieben; eine Beschreibung findet sich jedoch im PHP API.

- `int kto_check_blz(char *blz,char *kto)`
  (Beispiel)
- `int kto_check_pz(char *pz,char *kto,char *blz)`
- `int kto_check_blz_dbg(char *blz,char *kto,RETVAL *retvals)`
- `int kto_check_pz_dbg(char *pz,char *kto,char *blz,RETVAL *retvals)`
- `int kto_check(char *pz_or_blz,char *kto,char *lut_name)`
- `char *kto_check_str(char *pz_or_blz,char *kto,char *lut_name)`

5.6 Test von IBAN und strukturiertem Verwendungszweck

- `char *iban2bic(char *iban,int *retval)`
- `char *iban_gen(char *kto,char *blz,int *retval)`
- `int iban_check(char *iban,int *retval)`
- `int ipi_check(char *zweck)`
- `int ipi_gen(char *zweck,char *dst,char *papier)`

5.7 Abfrage von Bankdaten (aus der BLZ)

- `int lut_filialen(char *b,int *retval)`
- `char *lut_name(char *b,int zweigstelle,int *retval)`
  (Beispiel)
- `char *lut_name_kurz(char *b,int zweigstelle,int *retval)`
- `int lut_plz(char *b,int zweigstelle,int *retval)`
  (Beispiel)
- `char *lut_ort(char *b,int zweigstelle,int *retval)`
  (Beispiel)
- `int lut_pan(char *b,int zweigstelle,int *retval)`
- `char *lut_bic(char *b,int zweigstelle,int *retval)`
- `int lut_pz(char *b,int zweigstelle,int *retval)`
- `int lut_aenderung(char *b,int zweigstelle,int *retval)`
- `int lut_loeschung(char *b,int zweigstelle,int *retval)`
- `int lut_nachfolge_blz(char *b,int zweigstelle,int *retval)`
- `int lut_multiple(char *b,int *cnt,int **p_blz,char ***p_name,char ***p_name_kurz,int **p_plz, char ***p_ort,int **p_pan,char ***p_bic,int *p_pz,int **p_nr,char **p_aenderung,char **p_loeschung, int **p_nachfolge_blz,int *id,int *cnt_all,int **start_idx)`

5.8 Sonstige Funktionen

- `void lut_cleanup(void)`
  `int cleanup_kto(void)`

  Diese Funktionen sind synonym; sie geben den allokierten Speicher wieder frei. Eine dieser Funktionen sollte nach Benutzung der Bibliothek aufgerufen werden. (Beispiel)

- `int lut_valid(void)`

  Die Funktion lut_valid() testet, ob die geladene LUT-Datei im Gültigkeitszeitraum liegt; Rückgabewerte sind:

LUT2_NO_VALID_DATE	kein Gültigkeitsdatum in der Datei
LUT2_VALID	die Datei ist aktuell gültig
LUT2_NOT_YET_VALID	die Datei ist noch nicht gültig
LUT2_NO_LONGER_VALID	die Datei ist nicht mehr gültig

- `int lut_info(char *lut_name,char **info1,char **info2,int *valid1,int *valid2)`
- `int get_lut_info_b(char **info,char *lutname)`
- `int generate_lut(char *inputname,char *outputname,char *user_info,int lut_version)`
- `int generate_lut2(char *inputname,char *outputname,char *user_info,char *gueltigkeit, UINT4 *felder,int slots,int lut_version,UINT4 set)`
- `int generate_lut2_p(char *inputname,char *outputname,char *user_info,char *gueltigkeit, UINT4 felder,UINT4 filialen,int slots,int lut_version,int set)`
- `char *get_kto_check_version(void)`

6 Die PHP-Version von *konto_check*

6.1 Einführung

Es gibt zwei verschiedene Versionen der PHP-Bibliothek, die sich in der Aufrufkonvention unterscheiden. Sie werden hier ebenfalls als altes und neues API bezeichnet; diese Bezeichnungen haben jedoch mit den beiden APIs der C-Version nichts zu tun. Das PHP-Modul läßt sich mit der alten C-Bibliothek (bis 2.7) nicht gut anwenden, da einige grundlegende Funktionen und Locking-Mechanismen einfach fehlen.

Das Problem bei der PHP-Bibliothek war, daß viele Funktionen mehrere Rückgabewerte haben. In der ersten Version wurde dies so gelöst, daß ein Array mit den verschiedenen Werten zurückgegeben wurde. Die Funktionen sind dadurch jedoch etwas sperrig und unhandlich; es war das erste PHP-Modul, das ich schrieb, und ich kannte die Möglichkeiten der ZEND Engine einfach noch nicht.

In der neuen Version wird dagegen grundsätzlich mit skalaren Rückgabeparametern gearbeitet. In den Funktionen, die mehrere Rückgabeparameter haben, werden stattdessen Funktionsargumente benutzt, deren Aufrufkonvention als Call by Reference erzwungen wird (transparent für den Benutzer); diese Parameter sind in den meisten Fällen auch optional, so daß sie nur bei Bedarf angegeben werden müssen. Dadurch ergibt sich ein wesentlich anwenderfreundlicheres API als in der alten Version.

Die Reihenfolge der Aufrufparameter weicht bei einer Reihe Funktionen von der des C API ab. Das liegt daran, daß in C nur feste Parameterlisten verwendet werden; in PHP ist es allerdings manchmal geschickter, die Parameter in einer anderen Reihenfolge zu benutzen. So entspricht z.B. die PHP-Funktion `lut_ort($blz[,$retval[,$zweigstelle]])` der C-Funktion `lut_ort(blz,zweigstelle,retval)`; da man jedoch öfter den Rückgabewert der Funktion benötigt als eine Zweigstelle, wurden die Parameter vertauscht. Ansonsten müsste man immer die Zweigstelle mit angeben, auch wenn man sie eigentlich gar nicht braucht.

6.2 Installation der PHP-Version

Aktuell ist nur die ∗nix-Version von *konto_check* getestet; die Windows-Version wird in einer späteren Version folgen, sie benötigt noch einige Tests.

Die Installation ist relativ einfach: Die Datei php.zip (bzw. php.old.zip) ist auszupacken; sie generiert ein Verzeichnis php/, in dem alle benötigten Dateien zu finden sind. Wechseln Sie in dieses Verzeichnis und geben Sie dann die folgenden Befehle ein:

```
phpize
./configure
make
make install
```

(der letzte Schritt muß als Root erfolgen). Danach ist noch die Datei konto_check.ini in das Systemverzeichnis `/etc/php.d/` zu kopieren (unter Fedora/RedHat; bei anderen Distributionen liegt das Verzeichnis `php.d/` u.U. anderswo), und die Installation ist beendet. Mit `php -a` können Sie die PHP-Shell aufrufen und dann einige Tests machen. Anschließend muß noch der Webserver neu gestartet werden, und die Installation ist abgeschlossen.

7 Das alte PHP API

7.1 Initialisierung, Infos zur LUT-Datei, Speicherfreigabe

- `$ret=lut_init([$lut_name[,$required[,$set[,$used_lutfile]]]])`

  Diese Funktion dient zur einfachen Initialisierung der Bibliothek. Alle Parameter sind optional; für nicht übergebene Parameter werden die Standardwerte aus der INI-Datei bzw. *konto_check.h* genommen.

  Der Parameter `$lut_name` ist der Name der LUT-Datei; falls er nicht angegeben werden soll, kann auch ein Leerstring benutzt werden. In diesem Fall wird der Default-Wert der INI-Datei benutzt.

  Der Parameter `$required` ist in dieser Funktion ein skalarer Wert zwischen 0 und 9; die Zuordnung der mit den einzelnen Werten verknüpften Blocks ist im Anhang aufgeführt.

  Der Parameter `$set` ist ein Skalar mit dem Wert 0, 1 oder 2; er gibt den zu ladenden Datensatz der LUT-Datei an. Näheres dazu auch die C Funktion und in der Beschreibung des LUT-Dateiformats.

  (Beispiel) von lut_init()

- `$ret=kto_check_init([$lut_name[,$required[,$set[,$incremental[,$used_lutfile]]]]])`

  Diese Funktion ist weitgehend identisch mit lut_init; nur der Parameter `$incremental` ist dazugekommen, der angibt, ob inkrementell initialisiert werden soll oder nicht.

  Diese Funktion hat weniger Möglichkeiten als die des neuen Interfaces; insbesondere kann der Parameter `$required` nur ein *Skalar* sein, kein Array.

- `$ret=lut_valid()`

  Diese Funktion testet, ob der aktuell *geladene* Datensatz aktuell gültig ist. Mögliche Rückgabewerte sind:

5	LUT2_NO_VALID_DATE	kein Gültigkeitsdatum in der Datei
3	LUT2_VALID	die Datei ist aktuell gültig
-59	LUT2_NOT_YET_VALID	die Datei ist noch nicht gültig
-58	LUT2_NO_LONGER_VALID	die Datei ist nicht mehr gültig

  Zurückgegeben wird nur ein skalarer Wert (numerisch).

- `$ret=lut2_status()`

  Dies Funktion gibt ein Array zurück, in dem für jeden angeforderten Block der aktuelle Status zurückgegeben wird. Die Werte entsprechen der Variablen `$status` der Funktion `kto_check_init()`. Es werden sowohl symbolische als auch numerische Index-Werte für die Blocks gesetzt.

- `$ret=lut_free()`
  `$ret=lut_cleanup()`

  Diese Funktionen sind synonym. Sie geben allokierten Speicher wieder frei und löschen die entsprechenden Variablen, so daß die Bibliothek als nicht initialisiert gekennzeichnet ist. (Beispiel)

7.2 Konvertierung der numerischen Rückgabewerte in Klartext

Die folgenden Funktionen konvertieren die numerischen Rückgabewerte in Klartext in verschiedenen Formaten:

- `$ret=kto_check_retval2txt_short($ret)`

  Ausgabe als kurzer String, der den Makronamen aus *konto_check.h* entspricht.

- `$ret=kto_check_retval2txt($ret)`

  Ausgabe als Text, Kodierung der Umlaute mit ISO-8859-1.

- `$ret=kto_check_retval2html($ret)`

  Ausgabe als Text, Kodierung der Umlaute mit HTML Entities. (Beispiel)

- `$ret=kto_check_retval2dos($ret)`

  Ausgabe als Text, Kodierung der Umlaute mit cp850 (DOS).

- `$ret=kto_check_retval2utf8($ret)`

  Ausgabe als Text, Kodierung der Umlaute mit UTF-8.

7.3 Testfunktionen: BLZ/Prüfziffer und Konto

- `$retval=kto_check($blz,$kto[,$lut_name])`

  Diese Funktion testet, ob die BLZ/Konto Kombination gültig ist. Die Funktion gehört noch zu den alten C-Funktionen; falls die Bibliothek noch nicht initialisiert war, wird sie automatisch initialisiert. Der (optionale) Parameter `$lut_name` gibt den Dateinamen der LUT-Datei an; falls er nicht angegeben ist, werden die Standardwerte benutzt.

- `$retval=kto_check_str($blz,$kto[,$lut_name])`

  Diese Funktion entspricht der Funktion `kto_check()`; der Rückgabewert ist allerdings nicht wie bei dieser ein Skalar, sondern ein assoziatives Array mit den folgenden Elementen:

`retval['ret']`	Numerischer Rückgabewert
`retval['ret1']`	Rückgabewert in ISO-8859-1 Kodierung
`retval['ret2']`	Rückgabewert als kurzer String (Makroname)
`retval['ret3']`	Rückgabewert in HTML-Kodierung

- `$ret=kto_check_pz($pruefziffer,$kto[,$blz])` Diese Funktion testet eine Kontonummer mit einer Prüfziffermethode; mögliche Werte für die Prüfziffer sind 00 bis D3. Die Funktion setzt *nicht* voraus, daß die Bibliothek bereits initialisiert ist.

  Der Parameter `$blz` wird bei Prüfung mit den Methoden 52, 53, B6 und C0 benötigt; ansonsten kann er weggelassen werden.

- `$ret=kto_check_blz($blz,$kto)` Diese Funktion testet eine Kontonummer mit einer vorgegebenen Bankleitzahl. Die Funktion setzt voraus, daß die Bibliothek bereits initialisiert ist. (Beispiel)

7.4 Testfunktionen: IBAN, Strukturierter Verwendungszweck

Die Funktionen zu IBAN und Strukturiertem Verwendungszweck haben alle als Rückgabewert ein assoziatives Array. Bei den Funktionen wird – falls dies in *php_konto_check.h* eingeschaltet ist – das Funktionsargument ebenfalls zurückgegeben; dieser Parameter ist allerdings standardmäßig nicht gesetzt.

Im neuen API wird statt eines assoziativen Arrays in allen Fällen ein skalarer Wert zurückgegeben; die zusätzlichen Rückgabeparameter werden *by reference* zurückgegeben. Das Funktionsargument wird im neuen API grundsätzlich *nicht* zurückgegeben (es ist auch in im alten API nur noch aus Kompatibilitätsgründen vorhanden, nicht aus Überzeugung ;-)).

- `$ret=iban_check($iban)`

  Diese Funktion testet die IBAN auf Gültigkeit; falls es ein deutsches Konto ist, wird außerdem die enthaltene Kontonummer auf Gültigkeit geprüft. Zurückgegeben wird ein assoziatives Array mit den folgenden Elementen:

`retval['ret']`	Numerischer Rückgabewert
`retval['ret1']`	Rückgabewert in ISO-8859-1 Kodierung
`retval['ret2']`	Rückgabewert als kurzer String (Makroname)
`retval['ret3']`	Rückgabewert in HTML-Kodierung
`retval['ret_kto']`	Numerischer Rückgabewert der Konto-Prüfung
`retval['ret1_kto']`	Rückgabewert der Konto-Prüfung in ISO-8859-1 Kodierung
`retval['ret2_kto']`	Rückgabewert der Konto-Prüfung als kurzer String (Makroname)
`retval['ret3_kto']`	Rückgabewert der Konto-Prüfung in HTML-Kodierung
`retval['iban']`	das Argument `$iban` (nur bei USE_ARGS)

- `$ret=iban2bic($iban)`

  Diese Funktion bestimmt zu einer gegebenen IBAN den zugehörigen BIC (Bank Identifier Code). Die Funktion arbeitet nur mit deutschen Banken; außerdem muß natürlich der Block mit den BICs aus der LUT-Datei geladen sein. Es werden außerdem noch die BLZ und Kontonummer der vorgegebenen IBAN bestimmt. Die folgenden Werte werden zurückgegeben:

`retval['ret']`	Numerischer Rückgabewert
`retval['ret1']`	Rückgabewert in ISO-8859-1 Kodierung
`retval['ret2']`	Rückgabewert als kurzer String (Makroname)
`retval['ret3']`	Rückgabewert in HTML-Kodierung
`retval['bic']`	der gesuchte BIC
`retval['blz']`	die Bankleitzahl des IBAN
`retval['kto']`	die Kontonummer des IBAN
`retval['iban']`	das Argument `$iban` (nur bei USE_ARGS)

- `$ret=ipi_gen($zweck)` Diese Funktion generiert einen Strukturierten Verwendungszweck. Der Zweck darf nur Buchstaben und Zahlen enthalten; Buchstaben werden dabei in Großbuchstaben umgewandelt. Andere Zeichen sind hier nicht zulässig. Der Verwendungszweck wird links mit Nullen bis auf 18 Byte aufgefüllt, dann die Prüfsumme berechnet und eingesetzt. Die folgenden Werte werden zurückgegeben:

`retval['ret']`	Numerischer Rückgabewert
`retval['ret1']`	Rückgabewert in ISO-8859-1 Kodierung
`retval['ret2']`	Rückgabewert als kurzer String (Makroname)
`retval['ret3']`	Rückgabewert in HTML-Kodierung
`retval['zweck_out']`	der generierte Verwendungszweck
`retval['papier']`	Papierform des generierten Verwendungszwecks (mit Blanks)
`retval['zweck']`	der Parameter `$zweck` (nur bei USE_ARGS)

- `$ret=ipi_check($zweck)`

  Diese Funktion testet einen Strukturierten Verwendungszweck für eine IPI (International Payment Instruction). Der Zweck darf nur Buchstaben und Zahlen enthalten und muß genau 20 Byte lang sein, wobei eingestreute Blanks oder Tabs ignoriert werden. Die folgenden Werte werden zurückgegeben:

`retval['ret']`	Numerischer Rückgabewert
`retval['ret1']`	Rückgabewert in ISO-8859-1 Kodierung
`retval['ret2']`	Rückgabewert als kurzer String (Makroname)
`retval['ret3']`	Rückgabewert in HTML-Kodierung
`retval['zweck']`	der Parameter `$zweck` (nur bei USE_ARGS)

- `lut_cleanup()`
- `lut_filialen()`
- `lut_multiple()`
- `lut_info()`
- `lut_name()`
- `lut_name_kurz()`
- `lut_plz()`
- `lut_ort()`
- `lut_pan()`
- `lut_bic()`
- `lut_pz()`
- `lut_aenderung()`
- `lut_loeschung()`
- `lut_nachfolge_blz()`

8 Das neue PHP API

8.1 Einführung

In diesem Interface werden für die Rückgabe praktisch nur Skalare benutzt; sie können so direkt verwendet werden. Falls eine Funktion mehrere Werte zurückgibt, erfolgt dies über (i.A. optionale) Parameter. Diese Rückgabeparameter sind in der folgenden Liste mit `&$variablenname` gekennzeichnet, Parameter die *by value* übergeben werden, werden einfach als `$variablenname` angegeben. Die Übergabe *by reference* wird durch die Bibliothek erzwungen; es ist nicht notwendig, beim Funktionsaufruf den *call-by-reference* Operator `&` zu verwenden.

8.2 Initialisierung, Infos zur LUT-Datei etc., Speicherfreigabe

- `$ret=lut_init([$lut_name[,$required[,$set[,$used_lutfile]]]])`

  Diese Funktion dient zur einfachen Initialisierung der Bibliothek. Alle Parameter sind optional; für nicht übergebene Parameter werden die Standardwerte aus *konto_check.h* bzw. der INI-Datei genommen.

  Der Parameter `$lut_name` ist der Name der LUT-Datei; falls er nicht angegeben werden soll, kann auch ein Leerstring benutzt werden. In diesem Fall wird der Default-Wert der INI-Datei benutzt.

  Der Parameter `$required` ist ein skalarer Wert zwischen 0 und 9; die Zuordnung der mit den einzelnen Werten verknüpften Blocks ist im Anhang aufgeführt.

  Der Parameter `$set` ist ein Skalar mit dem Wert 0, 1 oder 2; er gibt den zu ladenden Datensatz der LUT-Datei an. Näheres dazu auch die C Funktion und in der Beschreibung des LUT-Dateiformats.

  (Beispiel) von lut_init()

- `$ret=kto_check_init([lut_name[,$required[,$set[,$incremental[,&$status[,$used_lutfile]]]]]])`

  Dies ist die allgemeine Initialisierungsfunktion der Bibliothek mit allen Optionen der C-Datei; im Gegensatz zur C-Version wird allerdings der Parameter `$status` hier als letzter Parameter übergeben, um die Kompatibilität zur alten Version zu wahren. Der Parameter `$required` kann in dieser Funktion auch ein Array sein; in dem Fall werden nur die angegebenen Blocks geladen.Falls der Parameter `$status` angegeben wird, wird in dieser Variablen ein assoziatives Array zurückgegeben, bei dem für jeden angeforderten Block der Status zurückgegeben wird.

- `$ret=lut_status()`
  `$ret=lut2_status(`

  Diese beiden Funktionen sind synonym; zurückgegeben wird ein Array, in dem für jeden angeforderten Block der aktuelle Status zurückgegeben wird. Die Werte entsprechen der Variablen `$status` der Funktion `kto_check_init()`.

- `$ret=lut_info([lut_name[,&$info1[,&$valid1[,&$info2[,&$valid2[,&$info3[,&$slots]]]]]]])`

  Diese Funktion liest aus einer LUT-Datei die Infoblocks und die Gültigkeitsdaten (falls vorhanden) und gibt sie in den entsprechenden Variablen zurück. Die Reihenfolge der Variablen entspricht nicht der C-Funktion; diese Reihenfolge ist hier günstiger, da die Variablen optional sind.

  Falls für den Parameter lut_name ein Leerstring übergeben wird, werden Infoblock und Gültigkeitszeitraum der *geladenen* LUT-Datei getestet.

  Falls die Variable `$info3` angegeben wird, werden in ihr einige statistische Daten zu der LUT-Datei sowie die in der Datei enthaltenen Blocktypen zurüückgeliefert.

  In der Variablen `$slots` wird ein Array zurückgeliefert, das für jeden Slot der LUT-Datei detaillierte Informationen enthält. Die Überschrift zu den Zeilen findet sich im Element 0.

- `$ret=read_lut_block($lut_name,$typ[,$retval])`

  Diese Funktion liest einen Block mit dem Typ `$typ` aus der angegebenen LUT-Datei. Der Rückgabestatus der Funktion läßt sich durch den optionalen Parameter `$retval` ermitteln.

- `$ret=write_lut_block($lut_name,$typ,$data)`

  Diese Funktion schreibt einen Block mit dem Typ `$typ` in die angegebenen LUT-Datei; die Daten befinden sich in der Variablen `$data`. Der Wert von `$typ` muß eine Integerzahl über 1000 sein; Werte bis 1000 sind für die *konto_check* Bibliothek reserviert.

- `$ret=copy_lutfile($old_name,$new_name,$slots)`
- `$ret=generate_lut2(inputname,outputname,[user_info[,gueltigkeit[,felder[,slots[,lut_version[,set]]]]]])`

  Diese Funktion dient zur Generierung einer LUT-Datei. Die Parameter entsprechen denen der C-Funktion. Alle Parameter außer inputname und outputname sind optional. Hier eine kurze Beschreibung:

  inputname: Dateiname der Bundesbank-Datei

  outputname: Dateiname der generierten LUT-Datei

  user_info: Info-Zeile für die Datei (Benutzer-Info)

  gueltigkeit: Gültigkeitszeitraum für den Datensatz. Der Gültigkeitszeitraum ist im Format JJJJMMTT-JJJJMMTT anzugeben, also z.B. 20090309-20090607 für den Zeitraum 9.3.2009 - 6.7.2009.

  felder: Diese Variable ist ein Skalar oder Array, in dem die in die Ausgabedatei aufzunehmenden Felder angegeben werden. Bei einem skalaren Wert sind die Felder im Anhang von 00liesmich.pdf aufgezählt; bei einem Array kann man explizit bestimmen, welche Felder in die Ausgabedatei aufgenommen werden sollen. Falls die LUT-Datei auch die Filialen enthalten soll, ist zusätzlich zu den Datenblocks noch der Wert LUT2_FILIALEN in das Array einzufügen; andernfalls werden nur die Hauptstellen berücksichtigt.

  Falls ein negativer skalarer Wert angegeben wird, werden die Werte des entsprechenden Levels mit den Filialen eingebunen.

  Falls der Parameter nicht angegeben wird: Level 5 ohne Filialen

  slots: Anzahl Slots im Inhaltsverzeichnis der LUT-Datei. Die Anzahl Slots kann später nicht mehr geändert werden; wenn alle Slots einer Datei belegt sind, kann kein zusätzlicher Block mehr geschrieben werden. Der Defaultwert ist 40.

  lut_version: Dateiversion der LUT-Datei, die geschrieben werden soll. Der Parameter kann die folgenden Werte annehmen:

  0: default (aktuelle Version, jetzt 2.0) 1: Version 1.0 (obsolet) 2: Version 1.1 (enthält nur BLZ und Prüfziffermethoden) 3: Version 2.0 (kann alle Felder enthalten)

  set: Datensatz, der geschrieben werden soll. Eine LUT-Datei kann zwei Datensätze mit unterschiedlichem Gültigkeitsdatum enthalten. Dieser Parameter legt fest, welcher Datensatz geschrieben werden soll (1 oder 2).

  Falls für set der Wert 0 angegeben wird, wird eine neue Datei angelegt und der Datensatz 1 benutzt; falls set>0 ist, wird der Datensatz an eine bestehende Datei angehängt.

- `$ret=lut_valid()`

  Diese Funktion testet, ob der aktuell *geladene* Datensatz aktuell gültig ist. Mögliche Rückgabewerte sind:

5	LUT2_NO_VALID_DATE	kein Gültigkeitsdatum in der Datei
3	LUT2_VALID	die Datei ist aktuell gültig
-59	LUT2_NOT_YET_VALID	die Datei ist noch nicht gültig
-58	LUT2_NO_LONGER_VALID	die Datei ist nicht mehr gültig

- `$ret=lut_free()`
  `$ret=lut_cleanup()`

  Diese Funktionen sind synonym. Sie geben allokierten Speicher wieder frei und löschen die entsprechenden Variablen, so daß die Bibliothek als nicht initialisiert gekennzeichnet ist. (Beispiel)

- `$ret=get_kto_check_version($typ)`

8.3 Konvertierung der numerischen Rückgabewerte in Klartext

Die folgenden Funktionen konvertieren die numerischen Rückgabewerte in Klartext in verschiedenen Formaten. Es gibt jeweils zwei Funktionsnamen; die kurze Variante ist als Funktionsalias zu der Langform definiert.

- `$ret_txt=kto_check_retval2txt_short($ret)`
  `$ret_txt=kc_short($ret)`

  Ausgabe als kurzer String, der den Makronamen aus *konto_check.h* entspricht.

- `$ret_txt=kto_check_retval2txt($ret)`
  `$ret_txt=kc_txt($ret)`

  Ausgabe als Text, Kodierung der Umlaute mit ISO-8859-1.

- `$ret_txt=kto_check_retval2html($ret)`
  `$ret_txt=kc_html($ret)`

  Ausgabe als Text, Kodierung der Umlaute mit HTML Entities. (Beispiel)

- `$ret_txt=kto_check_retval2dos($ret)`
  `$ret_txt=kc_dos($ret)`

  Ausgabe als Text, Kodierung der Umlaute mit cp850 (DOS).

- `$ret_txt=kto_check_retval2utf8($ret)`
  `$ret_txt=kc_utf8($ret)`

  Ausgabe als Text, Kodierung der Umlaute mit UTF-8.

8.4 Testfunktionen: BLZ/Prüfziffer und Konto

- `$ret=kto_check(blz,$kto[,$lut_name])`

  Diese Funktion stammt aus dem alten C API; sie dient zum Test eines Kontos. Falls die Bibliothek noch nicht initialisiert wurde, wird sie in dieser Funktion implizit initialisiert, entweder mit der Datei `$lut_name` oder Defaultwerten (falls der Parameter nicht angegeben ist). Bei der Initialisierung werden nur die Bankleitzahlen und Prüfziffermethoden geladen. Der Rückgabewert ist numerisch.

- `$ret=kto_check_str(blz,$kto[,$ret_type[,$lut_name]])`

  Diese Funktion entspricht der Funktion `kto_check()`; allerdings ist der Rückgabewert ein String. Diese Funktion ist *nicht* kompatibel zum alten API; in diesem war der dritte Parameter `$lut_name`, während im neuen API `$lut_name` als vierter Parameter übergeben wird.

  Der Typ der Rückgabe läßt sich über den dritten Parameter festlegen:

  1: Ausgabe als kurzer String
  2: Ausgabe als Text; Umlaute in ISO-8859-1
  3: Ausgabe als Text; Umlaute als HTML Entities
  4: Ausgabe als Text; Umlaute in cp850 (DOS)
  5: Ausgabe als Text; Umlaute in UTF-8
  (anderer Wert): Ausgabe als Text; Umlaute als HTML Entities

- `$ret=kto_check_pz($pz,$kto[,$blz])`

  Diese Funktion testet eine Kontonummer mit einer Prüfziffermethode; mögliche Werte für die Prüfziffer sind 00 bis D3. Die Funktion setzt *nicht* voraus, daß die Bibliothek bereits initialisiert ist.

  Der Parameter `$blz` wird bei Prüfung mit den Methoden 52, 53, B6 und C0 benötigt; ansonsten kann er weggelassen werden.

- `$ret=kto_check_blz($blz,$kto)`

  Diese Funktion testet eine Kontonummer mit einer vorgegebenen Bankleitzahl. Die Funktion setzt voraus, daß die Bibliothek bereits initialisiert ist. (Beispiel)

- `$ret=kto_check_pz_dbg($pz,$kto[,$blz[,&$methode[,&$pz_methode[,&$pz[,&$pz_pos]]]]])`
  `$ret=kto_check_blz_dbg($blz,$kto[,&$methode[,&$pz_methode[,&$pz[,&$pz_pos]]]])`

  Diese beiden Funktionen sind nur verfügbar, wenn die Bibliothek mit der DEBUG-Option kompiliert wurde. Sie entsprechen den Funktionen `kto_check_pz()` bzw. `kto_check_blz();` in den zusätzlichen Variablen werden jedoch noch einige Testparameter zurückgegeben:

  `$methode:` die verwendete Prüfziffermethode (alphanumerisch mit zwei oder drei Stellen, inklusive Untermethode, z.B. 90c)

  `$pz_methode:` ebenfalls die Prüfziffermethode, jedoch rein numerisch. Die Untermethode wird mit 1000 multipliziert und addiert (dies ist die interne Darstellung innerhalb der *konto_check* Bibliothek, z.B. 3090 für Methode 90c)

  `$pz:` die *berechnete* Prüfziffer. Falls keine Prüfung stattfindet, wird -1 zurückgegeben.

  `$pz_pos:` die Position der Prüfziffer (beginnend mit 0 für die 1. Stelle). Falls keine Prüfung stattfindet, wird für auch für diesen Wert -1 zurückgegeben.

  Bei der Funktion `kto_check_pz_dbg()` ist darauf zu achten, daß die BLZ vor den anderen Parametern übergeben werden muß; falls sie nicht benötigt wird, kann für den Parameter auch 0 oder ein Leerstring übergeben werden.

8.5 Testfunktionen: IBAN, Strukturierter Verwendungszweck

- `$ret=iban_check(iban[,&$ret_kto])`

  Diese Funktion testet die Gültigkeit einer IBAN. Falls der Parameter `$ret_kto` angegeben ist, wird für deutsche Bankverbindungen wird außerdem noch die Gültigkeit der Bankverbindung mit den normalen Testroutinen geprüft und das Ergebnis in diesem Parameter zurückgegeben.

- `$bic=iban2bic(iban[,&$ret[,&$blz[,&$kto]]])`

  Diese Funktion bestimmt zu einer gegebenen IBAN den BIC (Bank Identifier Code). Die Funktion arbeitet nur mit deutschen Banken; außerdem muß natürlich der Block mit den BICs aus der LUT-Datei geladen sein. Der Rückgabestatus der Funktion wird optional *by reference* in der Funktion `$ret` zurückgegeben; ebenfalls können auf Wunsch BLZ und Kontonummer aus der IBAN bestimmt werden.

- `$zweck_out=ipi_gen(zweck[,&$ret[,&$papier]])`

  Diese Funktion generiert einen Strukturierten Verwendungszweck. Der Zweck darf nur Buchstaben und Zahlen enthalten; Buchstaben werden dabei in Großbuchstaben umgewandelt. Andere Zeichen sind hier nicht zulässig. Der Verwendungszweck wird links mit Nullen bis auf 18 Byte aufgefüllt, dann die Prüfsumme berechnet und eingesetzt. Falls der Parameter `$ret` angegeben ist, wird in ihm der Funktionsstatus zurückgeliefert.

  Falls der Parameter `$papier` angegeben ist, wird in ihm die „Papierform“ des generierten Verwendungszwecks (mit jeweils einem Blank nach vier Zeichen) zurückgegeben.

- `$ret=ipi_check(zweck)`

  Diese Funktion testet einen Strukturierten Verwendungszweck für eine IPI (International Payment Instruction). Der Zweck darf nur Buchstaben und Zahlen enthalten und muß genau 20 Byte lang sein, wobei eingestreute Blanks oder Tabs ignoriert werden.

8.6 Abfrage von Bankdaten (Felder der Bundesbank-Datei)

Die folgenden Funktionen liefern die verschiedenen Felder der Bankleitzahlendatei zu einer vorgegebenen Bankleitzahl. Die Funktionen liefern das Ergebnis direkt zurück; der Funktionsstatus kann jeweils durch den (optionalen) Parameter `$retval` erhalten werden. Eine ausführliche Beschreibung der einzelnen Datensätze findet sich im Anhang.

Der Parameter `$zweigstelle` ist die Zweigstelle, zu der das entsprechende Feld bestimmt werden soll. Die Hauptstelle hat immer den Wert 0; die Anzahl der Filialen kann über die Funktion `lut_filialen()` bestimmt werden. Die zurückgelieferte Anzahl ist die Gesamtanzahl, inklusive der Hauptstelle. Wird ein zu hoher Wert angegeben, wird `$ret` auf die Fehlermeldung LUT2_INDEX_OUT_OF_RANGE gesetzt und NULL zurückgeliefert.

- `$filialen=lut_filialen($blz[,&$retval])`

  Diese Funktion bestimmt die Anzahl der Filialen (inklusive der Hauptstelle) einer Bank.

- `$bankname=lut_name($blz[,&$retval[,$zweigstelle]])`

  Diese Funktion bestimmt den (Lang-)Namen der Bank. (Beispiel)

- `$bankname_kurz=lut_name_kurz($blz[,&$retval[,$zweigstelle]])`

  Diese Funktion bestimmt den Kurzbezeichnung mit Ort der Bank.

- `$plz=lut_plz($blz[,&$retval[,$zweigstelle]])`

  Diese Funktion bestimmt die Postleitzahl der Bank. (Beispiel)

- `$ort=lut_ort($blz[,&$retval[,$zweigstelle]])`

  Diese Funktion bestimmt den Ort der Bank. (Beispiel)

- `$pan=lut_pan($blz[,&$retval[,$zweigstelle]])`

  Diese Funktion bestimmt die Institutsnummer für PAN der Bank.

- `$bic=lut_bic($blz[,&$retval[,$zweigstelle]])`

  Diese Funktion bestimmt den BIC (Bank Identifier Code) der Bank.

- `$nr=lut_nr($blz[,&$retval[,$zweigstelle]])`

  Diese Funktion bestimmt die Datensatznummer zu einer angegebenen Bank. Die Filialen haben jeweils eigene Datensatznummern.

- `$pruefziffer=lut_pz($blz[,&$retval[,$zweigstelle]])`

  Diese Funktion bestimmt das Prüfzifferverfahren (00 bis D3) der Bank.

- `$aenderung=lut_aenderung($blz[,&$retval[,$zweigstelle]])`

  Änderungskennzeichen: Seit dem letzten Abschluss der Bankleitzahlendatei neu hinzugekommene Datensätze werden mit „A“ (Addition), geänderte Datensätze mit „M“ (Modified), unveränderte Datensätze mit „U“ (Unchanged) gekennzeichnet. Gelöschte Datensätze werden mit „D“ (Deletion) gekennzeichnet und sind – als Hinweis – letztmalig in der Bankleitzahlendatei enthalten. Diese Datensätze sind ab dem Gültigkeitstermin der Bankleitzahlendatei im Zahlungsverkehr nicht mehr zu verwenden.

- `$loeschung=lut_loeschung($blz[,&$retval[,$zweigstelle]])`

  Diese Funktion bestimmt das Flag, ob eine BLZ zur Löschung vorgesehen ist. Das Feld enthält das Merkmal „0“ (keine Angabe) oder „1“ (Bankleitzahl ist zur Löschung vorgesehen).

- `$nachfolge_blz=lut_nachfolge_blz($blz[,&$retval[,$zweigstelle]])`

  Diese Funktion bestimmt die Nachfolge-BLZ für Banken, die bald aus der Datei gelöscht werden sollen.

- `$ret=lut_multiple($blz[,$zweigstelle[,&$cnt[,&$name[,&$name_kurz[,&$plz[,&$ort[,&$pan[,&$bic[,&$pz[,&$nr[,&$aenderung[,&$loeschung[,&$nachfolge_blz]]]]]]]]]]]])`

  Dies ist eine Universalfunktion, die mehrere Felder der LUT-Datei gleichzeitig bestimmt. Anders als die C-Version kopiert die PHP-Version die gewünschten Felder für *eine* Zweigstelle (in C werden nur Pointer auf die internen Arrays sowie die Anzahl der Datensätze zurückgegeben).

  Rückgabewerte:

`LUT2_NOT_INITIALIZED`	die Bibliothek wurde noch nicht initialisiert
`LUT2_PARTIAL_OK`	es wurden nicht alle angeforderten Blocks geladen
`OK`	alles ok

9 Die Perl-Version von *konto_check*

9.1 Einführung

Anmerkung: Dies ist momentan nur ein Platzhalter; hier wird später noch etwas mehr kommen.

In Perl ist es möglich, zu unterscheiden, ob die aufrufende Funktion als Rückgabewert ein Array oder einen skalaren Wert erwartet; entsprechend kann man die Rückgabe flexibel gestalten. Davon wird vor allem in den Funktionen für LUT2 viel Gebrauch gemacht. So wird z.B. in der Funktion lut_name, falls sie in skalarem Kontext aufgerufen wird, nur der Name der Bank zurückgegeben; falls sie jedoch in Arraykontext aufgerufen wird, wird eine Liste zurückgegeben, die den Banknamen, den Rückgabestatus der Funktion als Integerwert sowie als langen und kurzen Text enthält.

Ein Nachteil in Perl ist, daß sich Arrayvariablen nicht leicht übergeben lassen; alle Übergabevariablen werden in *eine* flache Liste konvertiert, und es ist bei Übergabe mehrerer Arrays für die aufgerufene Funktion nicht mehr möglich, zu sagen, welcher Parameter welchem Array zuzuordnen ist (oder auch nur, wo ein skalarer Parameter in der Liste war). Daher wurde im Perl-API beim Aufruf auf die Übergabe von Arrays komplett verzichtet.

Auch bei der Rückgabe enthält das C-API einige Funktionen, die Pointer auf (interne) Arrays zurückliefern wie z.B. lut_multiple(); dies läßt sich in Perl nicht gut implementieren. Diese Funktionen haben dann in den diversen APIs unterschiedliche Repräsentationen (dies betrifft die Funktionen lut_multiple(), generate_lut2(), kto_check_init() und dump_lutfile()). Für Perl (und auch andere Sprachen wie PHP oder VB) gibt es in der C-Bibliothek die Funktionen generate_lut2_p(), kto_check_init_p() und dump_lutfile_p(), die eine etwas andere Aufrufkonvention haben und dann in diesen Sprachen benutzt werden können.

9.2 Installation der Perl-Version

Die Perl-Version wurde von den CPAN Testern getestet; die deutsche Version von *konto_check* läßt sich auf vielen Rechnern und Plattformen ohne Probleme kompilieren, und auch die Tests ergeben keine Fehler. Lediglich die österreichische Version erzeugt unter S390 Linux noch einige Fehler. Die Probleme bei Windows lagen daran, daß keine *zlib* installiert ist.

Die Installation folgt der Standardmethode für die Pakete: Laden Sie die Datei Business-KontoCheck-N.N.tar.gz herunter, und geben Sie die folgenden Befehle ein:

```
tar xzf Business-KontoCheck-N.N.tar.gz
perl Makefile.PL
make
make test
make install
```

(der letzte Schritt muß als Root erfolgen). Danach ist die Installation abgeschlossen, und das Modul kann benutzt werden.

10 Die Windows-Version von *konto_check* (DLL, VBA)

Dies ist momentan nur ein Platzhalter; hier wird später noch etwas kommen.

11 Anhang: Diverse Tabellen

11.1 Felder der Bundesbankdatei bzw. der LUT-Datei

Die verschiedenen Felder der Bankleitzahlendatei bzw. LUT-Datei können mit den Funktionen `lut_*` abgefragt werden. Um die Beschreibungen nicht für jedes API zu wiederholen, sind sie hier gesammelt. Sie stammen aus der Bankleitzahlen-Richtlinie, die von der Deutschen Bundesbank veröffentlicht wird.

- `(keine Funktion)` **[Feld 1: Bankleitzahl]**

  Die Bankleitzahl dient der eindeutigen Identifizierung eines Kreditinstitutes.

- `(keine Funktion)` **[Feld 2: Merkmal, ob bankleitzahlführendes Kreditinstitut oder nicht]**

  Innerhalb der *konto_check* Bibliothek sind Hauptstellen *immer* die Zweigstelle mit dem Index 0, danach folgen die anderen Zweigstellen. In der Bundesbankdatei sind oft Filialen vor der Hauptstelle eingeordnet.

  Für jede gemeldete Bankleitzahl wird genau ein Datensatz mit dem Merkmal „1“ im Feld 2 der Bankleitzahlendatei angelegt. Diese Datensätze sind im Zahlungsverkehr zu verwenden.

  Sofern die gleiche Bankleitzahl an anderen Orten für weitere Filialen eines Kreditinstitutes verwendet wird, werden diese Datensätze mit dem Merkmal „2“ im Feld 2 versehen. Datensätze mit dem Merkmal „2“ dienen nicht dem Zahlungsverkehr (Ausnahme siehe II. Ziffer 2 Feld 7), sondern unterstützen die ortsbezogene Suche der Bankleitzahl eines Kreditinstitutes.

- `bankname=lut_name()` **[Feld 3: Bezeichnung des Kreditinstitutes]**

  Für die Bezeichnung ist die Firmierung lt. Registerauszug bzw. Gesetz oder Satzung maßgeblich. Notwendige Kürzungen müssen sinnwahrend erfolgen. Filial- oder geschäftsfeldbezogene Zusätze sind hinter die Firmierung zu setzen. Die Eintragung erfolgt ohne Rechtsform. Führt ein Kreditinstitut am selben Ort eine zweite Bankleitzahl für Geschäftsfelder mit bedeutendem Zahlungsverkehr, ist der Firmierung ein deutlich unterscheidender Zusatz hinzuzufügen.

  Im Rahmen von Fusionen ist es vorübergehend zulässig, ein Kreditinstitut an einem Ort mit zwei Bankleitzahlen in der Bankleitzahlendatei zu führen. Hierbei muss zur Unterscheidung die Bezeichnung des Kreditinstitutes im Datensatz der ¿alten¿ Bankleitzahl mit dem Zusatz „-alt-“ versehen werden.

- `plz=lut_plz()` **[Felder 4 und 5: Postleitzahl und Ort]**
  `ort=lut_ort()`

  Die Angaben dienen zusammen mit der Bezeichnung des Kreditinstitutes in Feld 3 der eindeutigen Identifizierung. Anzugeben sind die Postleitzahl und der Ort des Sitzes des Kreditinstitutes bzw. der Filiale, wobei die Postleitzahl der eindeutigen Identifizierung des Ortes und nicht der postalischen Adressierung dient. Maßgeblich ist das Postleitzahlenund Ortsverzeichnis (Datafactory Postalcode) der Deutschen Post AG.

- `bankname_kurz=lut_name_kurz()` **[Feld 6: Kurzbezeichnung des Kreditinstitutes mit Ort]**

  Kurzbezeichnung und Ort sollen für die Empfängerangaben auf Rechnungen und Formularen angegeben werden. Hierdurch wird eine eindeutige Zuordnung der eingereichten Zahlungsaufträge ermöglicht. Auf Grund der Regelungen in den Richtlinien beziehungsweise Zahlungsverkehrs-Abkommen der deutschen Kreditwirtschaft ist die Länge der Angaben für die Bezeichnung des Kreditinstituts begrenzt.

  Die Kurzbezeichnung wird von dem Antragsteller selbst bzw. seinem Zentralinstitut festgelegt. Grundlage ist die Firmierung des Kreditinstitutes. Sofern erforderlich, sind die im Anhang 3 der Bankleitzahlen-Richtlinie aufgeführten Abkürzungen zu verwenden. Die Eintragung erfolgt ohne Rechtsform. Führt ein Kreditinstitut am selben Ort eine zweite Bankleitzahl für Geschäftsfelder mit bedeutendem Zahlungsverkehr, ist der Firmierung ein deutlich unterscheidender Zusatz hinzuzufügen.

  Im Rahmen von Fusionen ist es vorübergehend zulässig, ein Kreditinstitut an einem Ort mit zwei Bankleitzahlen in der Bankleitzahlendatei zu führen. Hierbei muss zur Unterscheidung, die Kurzbezeichnung des Kreditinstitutes im Datensatz der „alten“ Bankleitzahl mit dem Zusatz „-alt-“ versehen werden.

- `pan=lut_pan()` **[Feld 7: Institutsnummer für PAN]**

  Für den internationalen Kartenzahlungsverkehr mittels Bankkundenkarten haben die Spitzenverbände des Kreditgewerbes und die Deutsche Bundesbank eine gesonderte Institutsnummerierung festgelegt; danach erhält das kartenausgebende Kreditinstitut eine fünfstellige Institutsnummer für PAN (= Primary Account Number). Diese setzt sich zusammen aus der Institutsgruppennummer (grundsätzlich = vierte Stelle der Bankleitzahl) und einer nachfolgenden vierstelligen, von den einzelnen Institutionen frei gewählten Nummer. Abweichend hiervon ist den Mitgliedsinstituten des Bundesverbandes deutscher Banken e.V. sowie den Stellen der Deutschen Bundesbank stets die Institutsgruppennummer „2“ zugewiesen worden. Zusätzliche Institutsnummer(n) für PAN Sofern ein Kreditinstitut weitere Institutsnummern für PAN zu einer Bankleitzahl führt, werden zu dem Datensatz mit dem Merkmal „1“ im Feld 2 unter dem gleichen Ort zusätzliche Datensätze mit dem Merkmal „2“ im Feld 2 aufgenommen. Die Datensätze sind bis auf das Merkmal im Feld 2 und die Institutsnummern für PAN im Feld 7 identisch.

- `bic=lut_bic()` **[Feld 8: Bank Identifier Code (BIC)]**

  Der Bank Identifier Code (BIC) besteht aus acht oder elf zusammenhängenden Stellen und setzt sich aus den Komponenten BANKCODE (4 Stellen), LÄNDERCODE (2 Stellen), ORTSCODE (2 Stellen) sowie ggf. einem FILIALCODE (3 Stellen) zusammen.

  Jedes Kreditinstitut führt grundsätzlich einen BIC je Bankleitzahl und teilt diesen der Deutschen Bundesbank mit. Ausnahmen hiervon können auf Antrag für Bankleitzahlen zugelassen werden, die im BIC-gestützten Zahlungsverkehr (grenzüberschreitender Zahlungsverkehr und inländischer Individualzahlungsverkehr) nicht verwendet werden. Vergabe

  Der BIC wird von SWIFT vergeben. Die Einreichungsfristen für das BIC-Directory von SWIFT erfordern eine frühzeitige Antragstellung bei SWIFT. Die Deutsche Bundesbank ist berechtigt, ihr auf anderem Wege, z. B. im Rahmen ihrer Kundenbeziehung oder über das BIC-Directory von SWIFT bekannt gewordene BIC der Kreditinstitute in die Bankleitzahlendatei zu übernehmen; sie informiert das betroffene Kreditinstitut über die Änderung mit einem Kontrollausdruck des Datensatzes der Bankleitzahlendatei. Neue BIC sind frühestens zum gleichen Gültigkeitstermin wie bei SWIFT in die Bankleitzahlendatei aufzunehmen. Bei SWIFT abgemeldete BIC sind spätestens zum gleichen Termin wie bei SWIFT in der Bankleitzahlendatei abzumelden. Anträge auf Zuteilung, Änderung oder Löschung von BIC sind zusammen mit dem entsprechenden Antrag zur Bankleitzahlendatei – ggf. über die zuständige Zentralstelle – an die Deutsche Bundesbank zu leiten, die den Antrag an SWIFT weiterreicht.

Hinweis
Es ist zu beachten, dass durch die zurzeit unterschiedlichen Aktualisierungsrhythmen der Bankleitzahlendatei (vierteljährlich, siehe II. Ziffer 3) und des BIC-Directory (monatlich zum ersten Samstag eines Monats) in der Bankleitzahlendatei ggf. neue gültige BIC nicht bzw. bereits gelöschte BIC noch enthalten sind. Bei Anträgen für das BIC-Directory zur Aufnahme beziehungsweise Löschung eines BIC sollten die Gültigkeitstermine der Bankleitzahlendatei beachtet werden, sofern es sich um einen BIC handelt, der im europäischen Zahlungsverkehr Anwendung findet und der in die Bankleitzahlendatei aufgenommen beziehungsweise dort gelöscht werden sollte.

- `pruefziffer=lut_pz()` **[Feld 9: Kennzeichen für Prüfzifferberechnungsmethode]**

  Kreditinstitute sind verpflichtet, zum Zahlungsverkehr ausschließlich prüfziffergesicherte Kontonummern gemäß ihrer in der Bankleitzahlendatei angegebenen Prüfzifferberechnungsmethode zu verwenden. Die Angabe der Prüfzifferberechnungsmethode „09“ (keine Prüfzifferberechnung) ist zulässig.

  Datensätze, die in Feld 2 der Bankleitzahlendatei mit dem Merkmal „2“ versehen sind, erhalten dasselbe Kennzeichen zugeordnet, wie der in Feld 2 mit „1“ gekennzeichnete Datensatz derselben Bankleitzahl.

  Vergabe
  Die Deutsche Bundesbank führt eine Übersicht der im Kreditgewerbe angewandten Prüfzifferberechnungsmethoden. Die Vergabe neuer Kennzeichen für Prüfzifferberechnungsmethoden wird von der Deutschen Bundesbank zentral für die gesamte Kreditwirtschaft vorgenommen. Das Kennzeichen kann aus Buchstaben und Ziffern in beliebiger Kombination mit der Ausnahme des Buchstabens „O“ bestehen. Wird ein Verfahren gewählt, dem bisher kein Kennzeichen zugeordnet war, ist eine Mitteilung an die Deutsche Bundesbank ggf. über die jeweilige Zentralstelle (oder den jeweiligen Spitzenverband) notwendig, die eine exakte Beschreibung des Verfahrens, ein Rechenbeispiel und sowohl richtige als auch falsche Testkontonummern enthält. Soll die Beschreibung einer Prüfzifferberechnungsmethode geändert werden, so ist es erforderlich, dass sich alle Kreditinstitute, die diese Methode verwenden, abstimmen und die Deutsche Bundesbank informieren. Für die Einführung bzw. Änderung von Prüfziffernberechnungsmethoden sollte eine Vorlaufzeit von mindestens einem halben Jahr berücksichtigt werden.

  Die Einführung bzw. Änderung von Prüfzifferberechnungsmethoden sowie deren Gültigkeitstermin wird ebenso wie die Übersicht mit den Beschreibungen der Prüfzifferberechnungsmethoden auf der Internetseite der Deutschen Bundesbank sowie durch Veröffentlichungen der Spitzenverbände des Kreditgewerbes bekannt gegeben. Anfragen zu Beschreibungen von Prüfzifferberechnungsmethoden sind an die Kreditinstitute zu richten, die diese verwenden.

- `nr=lut_nr()` **[Feld 10: Nummer des Datensatzes]**

  Bei jeder Neuanlage eines Datensatzes wird automatisiert eine eindeutige Nummer vergeben. Eine einmal verwendete Nummer wird nicht noch einmal vergeben.

- `aenderung=lut_aenderung()` **[Feld 11: Änderungskennzeichen]**

  Seit dem letzten Abschluss der Bankleitzahlendatei neu hinzugekommene Datensätze werden mit „A“ (Addition), geänderte Datensätze mit „M“ (Modified), unveränderte Datensätze mit „U“ (Unchanged) gekennzeichnet. Gelöschte Datensätze werden mit „D“ (Deletion) gekennzeichnet und sind – als Hinweis – letztmalig in der Bankleitzahlendatei enthalten. Diese Datensätze sind ab dem Gültigkeitstermin der Bankleitzahlendatei im Zahlungsverkehr nicht mehr zu verwenden.

- `loeschung=lut_loeschung()` **[Feld 12: Hinweis auf eine beabsichtigte Bankleitzahllöschung]**

Zur frühzeitigen Information der Teilnehmer am Zahlungsverkehr und zur Beschleunigung der Umstellung der Bankverbindung kann ein Kreditinstitut, das die Löschung einer Bankleitzahl mit dem Merkmal „1“ im Feld 2 beabsichtigt, die Löschung ankündigen. Die Ankündigung kann erfolgen, sobald das Kreditinstitut seine Kunden über die geänderte Kontoverbindung informiert hat. Es wird empfohlen, diese Ankündigung mindestens eine Änderungsperiode vor der eigentlichen Löschung anzuzeigen.

Das Feld enthält das Merkmal „0“ (keine Angabe) oder „1“ (Bankleitzahl im Feld 1 ist zur Löschung vorgesehen).

Hinweise

- Die Löschung einer Bankleitzahl kann auch ohne eine vorherige Ankündigung der beabsichtigten Löschung vorgenommen werden.
- Die Ankündigung der beabsichtigten Löschung einer Bankleitzahl dient nur als Hinweis und darf nicht zur vorzeitigen Löschung der Bankleitzahl führen; die Bankleitzahl ist bis zu ihrer endgültigen Löschung weiterhin im Zahlungsverkehr zu verwenden.

- `nachfolge_blz=lut_nachfolge_blz()` **[Feld 13: Hinweis auf Nachfolge-Bankleitzahl]**

  Das Feld enthält entweder den Wert „00000000“ (Bankleitzahl ist nicht zur Löschung vorgesehen bzw. das Kreditinstitut hat keine Nachfolge-Bankleitzahl veröffentlicht) oder die Angabe einer „Bankleitzahl“. Eine Bankleitzahl kann angegeben sein, wenn das Feld 2 das Merkmal „1“ enthält und entweder die bevorstehende Löschung der Bankleitzahl angekündigt wurde (Feld 12 = „1“) oder die Bankleitzahl zum aktuellen Gültigkeitstermin gelöscht wird (Feld 11 = „D“).

  Ein Kreditinstitut kann die Veröffentlichung einer Nachfolge-Bankleitzahl veranlassen, sofern

  - Kontonummern in Verbindung mit der alten und neuen Bankleitzahl nicht doppelt vergeben sind und
  - die Prüfzifferberechnungsmethoden beider Bankleitzahlen gleich oder
  - die Prüfzifferberechnungsmethode der Bankleitzahl des übernehmenden Kreditinstitutes so gestaltet wird, dass alle Konten zur alten Bankleitzahl auch nach der neuen Prüfzifferberechnungsmethode richtig sind.

  Die Angabe einer Nachfolge-Bankleitzahl ist unwiderruflich.

  Auf Grund der Veröffentlichung einer Nachfolge-Bankleitzahl können Anwender diese in Zahlungsverkehrsdateien verwenden. Dazu wird in den Kontostammdaten – unter Beibehaltung der Kontonummer – die zur Löschung angekündigte Bankleitzahl bzw. die gelöschte Bankleitzahl im Feld 1 der Bankleitzahlendatei durch die Nachfolge-Bankleitzahl dauerhaft ersetzt. Kreditinstitute sind – wie bisher – nicht berechtigt, in Zahlungsverkehrsdateien Bankleitzahlen durch Nachfolge-Bankleitzahlen zu ersetzen

11.2 Namen der Datenblocks

Hier ist eine Liste der definierten Blocktypen und zugehörigen symbolischen Namen bzw. Makros:

Num.	Symbol/Makro	Beschreibung
0	(keins)	Abschluß der Liste *(notwendig)*
1	BLZ	BLZ
2	FILIALEN	Anzahl Fil.
3	NAME	Name

Num.	Symbol/Makro	Beschreibung
4	PLZ	Plz
5	ORT	Ort
6	NAME_KURZ	Name (kurz)
7	PAN	PAN
8	BIC	BIC
9	PZ	Pruefziffer
10	NR	Lfd. Nr.
11	AENDERUNG	Aenderung
12	LOESCHUNG	Loeschung
13	NACHFOLGE_BLZ	NachfolgeBLZ
14	NAME_NAME_KURZ	Name, Kurzn.
15	INFO	Infoblock

Für die Banknamen (Lang- und Kurzform) sind zwei verschiedene Varianten zur Speicherung möglich: Lang- und Kurzname können in einem Block zusammengefaßt werden, was zu einer kleineren Gesamtdatei führt (der Lang- und Kurzname jeweils einer Bank wird dabei als ein Datensatz geschrieben; aufgrund der räumlichen Nähe lassen sich die Daten so besser komprimieren), oder sie können als zwei getrennte Blocks geschrieben werden. Diese Variante führt zu einer größeren LUT-Datei, benötigt beim Entpacken jedoch weniger Speicher und ist vor allem vorteilhaft, wenn nur einer der beiden Blocks benötigt wird.

11.3 Datenblocks für verschiedene (skalare) Werte von `required`

Der Parameter required (Integerzahl) gibt die zu ladenden Datenblocks an; er kann die Werte 0...9 annehmen. Es werden immer die Datenblocks von BLZ, Prüfziffer und Filialen sowie die folgenden zusätzlichen Datenblocks geladen:

0: (keine zusätzlichen)
1: Kurzname
2: Kurzname, BIC
3: Name, PLZ, Ort
4: Name, PLZ, Ort, BIC
5: Name+Kurzname, PLZ, Ort, BIC
6: Name+Kurzname, PLZ, Ort, BIC, Nachfolge-BLZ
7: Name+Kurzname, PLZ, Ort, BIC, Nachfolge-BLZ, Änderungsdatum
8: Name+Kurzname, PLZ, Ort, BIC, Nachfolge-BLZ, Änderungsdatum, Löschdatum
9: Name+Kurzname, PLZ, Ort, BIC, Nachfolge-BLZ, Änderungsdatum, Löschdatum, PAN, Laufende Nr. des Datensatzes (kompletter Datensatz)

Mit der Bezeichnung Name+Kurzname ist der Blocktyp 14 bezeichnet; auch falls nur der Name oder Kurzname angefordert wird, werden beide geladen, da sie in einem gemeinsamen Block gespeichert sind.

11.4 Rückgabewerte der Bibliothek

Hier sind die Rückgabewerte der verschiedenen Funktionen. In den verschiedenen Versionen gibt Funktionen, die diese Umwandlung machen: PHP(alt), PHP(neu), C

11.5 Klartextbeschreibung der Rückgabewerte

In dieser Tabelle sind die Makronamen in Klammern mit angegeben; unten sind sie alphabetisch sortiert noch einmal aufgeführt.

-77 (BAV_FALSE) BAV denkt, das Konto ist falsch (konto_check hält es für richtig)

-76 (LUT2_NO_USER_BLOCK) User-Blocks müssen einen Typ > 1000 haben
-75 (INVALID_SET) für ein LUT-Set sind nur die Werte 0, 1 oder 2 möglich
-74 (NO_GERMAN_BIC) Ein Konto kann kann nur für deutsche Banken geprüft werden
-73 (IPI_CHECK_INVALID_LENGTH) Der zu validierende strukturierete Verwendungszweck muß genau 20 Zeichen enthalten
-72 (IPI_INVALID_CHARACTER) Im strukturierten Verwendungszweck dürfen nur alphanumerische Zeichen vorkommen
-71 (IPI_INVALID_LENGTH) Die Länge des IPI-Verwendungszwecks darf maximal 18 Byte sein
-70 (LUT1_FILE_USED) Es wurde eine LUT-Datei im Format 1.0/1.1 geladen
-69 (MISSING_PARAMETER) Bei der Kontoprüfung fehlt ein notwendiger Parameter (BLZ oder Konto)
-68 (IBAN2BIC_ONLY_GERMAN) Die Funktion iban2bic() arbeitet nur mit deutschen Bankleitzahlen
-67 (IBAN_OK_KTO_NOT) Die Prüfziffer der IBAN stimmt, die der Kontonummer nicht
-66 (KTO_OK_IBAN_NOT) Die Prüfziffer der Kontonummer stimmt, die der IBAN nicht
-65 (TOO_MANY_SLOTS) Es sind nur maximal 500 Slots pro LUT-Datei möglich (Neukompilieren erforderlich)
-64 (INIT_FATAL_ERROR) Initialisierung fehlgeschlagen (init_wait geblockt)
-63 (INCREMENTAL_INIT_NEEDS_INFO) Ein inkrementelles Initialisieren benötigt einen Info-Block in der LUT-Datei
-62 (INCREMENTAL_INIT_FROM_DIFFERENT_FILE) Ein inkrementelles Initialisieren mit einer anderen LUT-Datei ist nicht möglich
-61 (DEBUG_ONLY_FUNCTION) Die Funktion ist nur in der Debug-Version vorhanden
-60 (LUT2_INVALID) Kein Datensatz der LUT-Datei ist aktuell gültig
-59 (LUT2_NOT_YET_VALID) Der Datensatz ist noch nicht gültig
-58 (LUT2_NO_LONGER_VALID) Der Datensatz ist nicht mehr gültig
-57 (LUT2_GUELTIGKEIT_SWAPPED) Im Gültigkeitsdatum sind Anfangs- und Enddatum vertauscht
-56 (LUT2_INVALID_GUELTIGKEIT) Das angegebene Gültigkeitsdatum ist ungültig (Soll: JJJJMMTT-JJJJMMTT)
-55 (LUT2_INDEX_OUT_OF_RANGE) Der Index für die Filiale ist ungültig
-54 (LUT2_INIT_IN_PROGRESS) Die Bibliothek wird gerade neu initialisiert
-53 (LUT2_BLZ_NOT_INITIALIZED) Das Feld BLZ wurde nicht initialisiert
-52 (LUT2_FILIALEN_NOT_INITIALIZED) Das Feld Filialen wurde nicht initialisiert
-51 (LUT2_NAME_NOT_INITIALIZED) Das Feld Bankname wurde nicht initialisiert
-50 (LUT2_PLZ_NOT_INITIALIZED) Das Feld PLZ wurde nicht initialisiert
-49 (LUT2_ORT_NOT_INITIALIZED) Das Feld Ort wurde nicht initialisiert
-48 (LUT2_NAME_KURZ_NOT_INITIALIZED) Das Feld Kurzname wurde nicht initialisiert
-47 (LUT2_PAN_NOT_INITIALIZED) Das Feld PAN wurde nicht initialisiert
-46 (LUT2_BIC_NOT_INITIALIZED) Das Feld BIC wurde nicht initialisiert
-45 (LUT2_PZ_NOT_INITIALIZED) Das Feld Prüfziffer wurde nicht initialisiert
-44 (LUT2_NR_NOT_INITIALIZED) Das Feld NR wurde nicht initialisiert
-43 (LUT2_AENDERUNG_NOT_INITIALIZED) Das Feld Änderung wurde nicht initialisiert
-42 (LUT2_LOESCHUNG_NOT_INITIALIZED) Das Feld Löschung wurde nicht initialisiert
-41 (LUT2_NACHFOLGE_BLZ_NOT_INITIALIZED) Das Feld Nachfolge-BLZ wurde nicht initialisiert
-40 (LUT2_NOT_INITIALIZED) die Programmbibliothek wurde noch nicht initialisiert
-39 (LUT2_FILIALEN_MISSING) der Block mit der Filialenanzahl fehlt in der LUT-Datei
-38 (LUT2_PARTIAL_OK) es wurden nicht alle Blocks geladen

-37 (LUT2_Z_BUF_ERROR) Buffer error in den ZLIB Routinen
-36 (LUT2_Z_MEM_ERROR) Memory error in den ZLIB-Routinen
-35 (LUT2_Z_DATA_ERROR) Datenfehler im komprimierten LUT-Block
-34 (LUT2_BLOCK_NOT_IN_FILE) Der Block ist nicht in der LUT-Datei enthalten
-33 (LUT2_DECOMPRESS_ERROR) Fehler beim Dekomprimieren eines LUT-Blocks
-32 (LUT2_COMPRESS_ERROR) Fehler beim Komprimieren eines LUT-Blocks
-31 (LUT2_FILE_CORRUPTED) Die LUT-Datei ist korrumpiert
-30 (LUT2_NO_SLOT_FREE) Im Inhaltsverzeichnis der LUT-Datei ist kein Slot mehr frei
-29 (UNDEFINED_SUBMETHOD) Die (Unter)Methode ist nicht definiert
-28 (EXCLUDED_AT_COMPILETIME) Der benötigte Programmteil wurde beim Kompilieren deaktiviert
-27 (INVALID_LUT_VERSION) Die Versionsnummer für die LUT-Datei ist ungültig
-26 (INVALID_PARAMETER_STELLE1) ungültiger Prüfparameter (erste zu prüfende Stelle)
-25 (INVALID_PARAMETER_COUNT) ungültiger Prüfparameter (Anzahl zu prüfender Stellen)
-24 (INVALID_PARAMETER_PRUEFZIFFER) ungültiger Prüfparameter (Position der Prüfziffer)
-23 (INVALID_PARAMETER_WICHTUNG) ungültiger Prüfparameter (Wichtung)
-22 (INVALID_PARAMETER_METHODE) ungültiger Prüfparameter (Rechenmethode)
-21 (LIBRARY_INIT_ERROR) Problem beim Initialisieren der globalen Variablen
-20 (LUT_CRC_ERROR) Prüfsummenfehler in der blz.lut Datei
-19 (FALSE_GELOESCHT) falsch (die BLZ wurde außerdem gelöscht)
-18 (OK_NO_CHK_GELOESCHT) ok, ohne Prüfung (die BLZ wurde allerdings gelöscht)
-17 (OK_GELOESCHT) ok (die BLZ wurde allerdings gelöscht)
-16 (BLZ_GELOESCHT) die Bankleitzahl wurde gelöscht
-15 (INVALID_BLZ_FILE) Fehler in der blz.txt Datei (falsche Zeilenlänge)
-14 (LIBRARY_IS_NOT_THREAD_SAFE) undefinierte Funktion; die library wurde mit THREAD_SAFE=0 kompiliert
-13 (FATAL_ERROR) schwerer Fehler im Konto_check-Modul
-12 (INVALID_KTO_LENGTH) ein Konto muß zwischen 1 und 10 Stellen haben
-11 (FILE_WRITE_ERROR) kann Datei nicht schreiben
-10 (FILE_READ_ERROR) kann Datei nicht lesen
 -9 (ERROR_MALLOC) kann keinen Speicher allokieren
 -8 (NO_BLZ_FILE) die blz.txt Datei wurde nicht gefunden
 -7 (INVALID_LUT_FILE) die blz.lut Datei ist inkosistent/ungültig
 -6 (NO_LUT_FILE) die blz.lut Datei wurde nicht gefunden
 -5 (INVALID_BLZ_LENGTH) die Bankleitzahl ist nicht achtstellig
 -4 (INVALID_BLZ) die Bankleitzahl ist ungültig
 -3 (INVALID_KTO) das Konto ist ungültig
 -2 (NOT_IMPLEMENTED) die Methode wurde noch nicht implementiert
 -1 (NOT_DEFINED) die Methode ist nicht definiert
  0 (FALSE) falsch
  1 (OK) ok
  2 (OK_NO_CHK) ok, ohne Prüfung
  3 (OK_TEST_BLZ_USED) ok; für den Test wurde eine Test-BLZ verwendet
  4 (LUT2_VALID) Der Datensatz ist aktuell gültig
  5 (LUT2_NO_VALID_DATE) Der Datensatz enthält kein Gültigkeitsdatum
  6 (LUT1_SET_LOADED) Die Datei ist im alten LUT-Format (1.0/1.1)
  7 (LUT1_FILE_GENERATED) ok; es wurde allerdings eine LUT-Datei im alten Format (1.0/1.1) generiert

11.6 Makronamen bzw. Kurznamen der Rückgabewerte (alphabetisch sortiert)

Die folgenden Makronamen (in C) bzw. Konstanten (in PHP, falls SYMBOLIC_RETVALS in *konto_check.h* gesetzt ist) sind definiert:

-77	BAV_FALSE
-16	BLZ_GELOESCHT
-61	DEBUG_ONLY_FUNCTION
-9	ERROR_MALLOC
-28	EXCLUDED_AT_COMPILETIME
0	FALSE
-19	FALSE_GELOESCHT
-13	FATAL_ERROR
-10	FILE_READ_ERROR
-11	FILE_WRITE_ERROR
-67	IBAN_OK_KTO_NOT
-68	IBAN2BIC_ONLY_GERMAN
-62	INCREMENTAL_INIT_FROM_DIFFERENT_FILE
-63	INCREMENTAL_INIT_NEEDS_INFO
-64	INIT_FATAL_ERROR
-4	INVALID_BLZ
-15	INVALID_BLZ_FILE
-5	INVALID_BLZ_LENGTH
-3	INVALID_KTO
-12	INVALID_KTO_LENGTH
-7	INVALID_LUT_FILE
-27	INVALID_LUT_VERSION
-25	INVALID_PARAMETER_COUNT
-22	INVALID_PARAMETER_METHODE
-24	INVALID_PARAMETER_PRUEFZIFFER
-26	INVALID_PARAMETER_STELLE1
-23	INVALID_PARAMETER_WICHTUNG
-75	INVALID_SET
-73	IPI_CHECK_INVALID_LENGTH
-72	IPI_INVALID_CHARACTER
-71	IPI_INVALID_LENGTH
-66	KTO_OK_IBAN_NOT
-21	LIBRARY_INIT_ERROR
-14	LIBRARY_IS_NOT_THREAD_SAFE
-20	LUT_CRC_ERROR
7	LUT1_FILE_GENERATED
-70	LUT1_FILE_USED
6	LUT1_SET_LOADED
-43	LUT2_AENDERUNG_NOT_INITIALIZED
-46	LUT2_BIC_NOT_INITIALIZED
-34	LUT2_BLOCK_NOT_IN_FILE
-53	LUT2_BLZ_NOT_INITIALIZED
-32	LUT2_COMPRESS_ERROR
-33	LUT2_DECOMPRESS_ERROR
-31	LUT2_FILE_CORRUPTED
-39	LUT2_FILIALEN_MISSING
-52	LUT2_FILIALEN_NOT_INITIALIZED
-57	LUT2_GUELTIGKEIT_SWAPPED

-55	LUT2_INDEX_OUT_OF_RANGE
-54	LUT2_INIT_IN_PROGRESS
-60	LUT2_INVALID
-56	LUT2_INVALID_GUELTIGKEIT
-42	LUT2_LOESCHUNG_NOT_INITIALIZED
-41	LUT2_NACHFOLGE_BLZ_NOT_INITIALIZED
-48	LUT2_NAME_KURZ_NOT_INITIALIZED
-51	LUT2_NAME_NOT_INITIALIZED
-58	LUT2_NO_LONGER_VALID
-30	LUT2_NO_SLOT_FREE
-76	LUT2_NO_USER_BLOCK
5	LUT2_NO_VALID_DATE
-40	LUT2_NOT_INITIALIZED
-59	LUT2_NOT_YET_VALID
-44	LUT2_NR_NOT_INITIALIZED
-49	LUT2_ORT_NOT_INITIALIZED
-47	LUT2_PAN_NOT_INITIALIZED
-38	LUT2_PARTIAL_OK
-50	LUT2_PLZ_NOT_INITIALIZED
-45	LUT2_PZ_NOT_INITIALIZED
4	LUT2_VALID
-37	LUT2_Z_BUF_ERROR
-35	LUT2_Z_DATA_ERROR
-36	LUT2_Z_MEM_ERROR
-69	MISSING_PARAMETER
-8	NO_BLZ_FILE
-74	NO_GERMAN_BIC
-6	NO_LUT_FILE
-1	NOT_DEFINED
-2	NOT_IMPLEMENTED
1	OK
-17	OK_GELOESCHT
2	OK_NO_CHK
-18	OK_NO_CHK_GELOESCHT
3	OK_TEST_BLZ_USED
-65	TOO_MANY_SLOTS
-29	UNDEFINED_SUBMETHOD

12 VERSIONEN

Datum	Version
01.05.02	Version 0.1
13.06.02	Version 0.2
10.07.02	Version 0.3
13.09.02	Version 1.0
10.10.02	Version 1.0.1
06.11.02	Version 1.0.2
04.02.03	Version 1.0.3
13.03.03	Version 1.1.0
16.04.03	Version 1.1.1
25.06.03	Version 1.1.2
16.01.04	Version 1.1.3

Datum	Version
12.10.04	Version 1.1.4
16.12.04	Version 1.1.5
16.01.04	Version 2.0-Alpha-1
12.10.04	Version 2.0-Beta-1
16.12.04	Version 2.0-Beta-2
06.08.05	Version 2.0 final
01.12.05	Version 2.0.1
26.05.06	Version 2.0.2
23.08.06	Version 2.0.3
20.11.06	Version 2.0.4
13.03.07	Version 2.0.5
26.05.07	Version 2.1
21.08.07	Version 2.2
25.08.07	Version 2.3 (nur für Perl, Bugfix release)
13.11.07	Version 2.4
16.02.08	Version 2.5
10.04.08	Version 2.6
23.08.08	Version 2.7 (parallel mit 2.92)
23.04.08	Version 2.91
23.08.08	Version 2.92
08.09.08	Version 2.93
13.01.09	Version 2.94/2.95 (bei 2.94 war ein kleiner Fehler in den Perl-Testroutinen)
02.03.09	Version 2.96
08.03.09	Version 2.97 (in 2.96 war ein fataler Fehler in den PHP-Routinen)
09.05.09	Version 2.98

13 Copyright

Allgemeiner Index

Felder der Bundesbankdatei, 23

Installation: Perl-Version, 22
Installation: PHP-Version, 13

lut_aenderung, 25
lut_bic, 24
lut_loeschung, 25
lut_nachfolge_blz, 26
lut_name, 23
lut_name_kurz, 24
lut_nr, 25
lut_ort, 23
lut_pan, 24
lut_plz, 23
lut_pz, 25

numerische Rückgabewerte, 27

required (skalar): Datenblocks, 27

Index der C-Funktionen

cleanup_kto, 12

generate_lut, 12
generate_lut2, 12
generate_lut2_p, 12
get_kto_check_version, 12
get_lut_info_b, 12

iban2bic, 11
iban_check, 11
iban_gen, 11
ipi_check, 11
ipi_gen, 11

kto_check, 11
kto_check_blz, 11
kto_check_blz_dbg, 11
kto_check_init, 10
kto_check_init_p, 9
kto_check_pz, 11
kto_check_pz_dbg, 11
kto_check_retval2dos, 10
kto_check_retval2html, 10
kto_check_retval2txt, 10
kto_check_retval2txt_short, 10
kto_check_retval2utf8, 10
kto_check_str, 11

lut_aenderung, 11
lut_bic, 11
lut_cleanup, 12
lut_filialen, 11
lut_info, 12
lut_init, 9
lut_loeschung, 11
lut_multiple, 11
lut_nachfolge_blz, 11
lut_name, 11
lut_name_kurz, 11
lut_ort, 11
lut_pan, 11
lut_plz, 11
lut_pz, 11
lut_valid, 12

Index der PHP-Funktionen (alt)

iban2bic, 15
iban_check, 15
ipi_check, 16
ipi_gen, 15

kto_check, 14
kto_check_blz, 15
kto_check_init, 13
kto_check_pz, 14
kto_check_retval2dos, 14
kto_check_retval2html, 14
kto_check_retval2txt, 14
kto_check_retval2txt_short, 14
kto_check_retval2utf8, 14
kto_check_str, 14

lut2_status, 14
lut_aenderung, 16
lut_bic, 16
lut_cleanup, 14, 16
lut_filialen, 16
lut_free, 14
lut_info, 16
lut_init, 13
lut_loeschung, 16
lut_multiple, 16
lut_nachfolge_blz, 16
lut_name, 16
lut_name_kurz, 16
lut_ort, 16
lut_pan, 16
lut_plz, 16
lut_pz, 16
lut_valid, 13

Index der PHP-Funktionen (neu)

copy_lutfile, 18

generate_lut2, 18
get_kto_check_version, 18

iban2bic, 20
iban_check, 20
ipi_check, 20
ipi_gen, 20

kc_dos, 19
kc_html, 19
kc_short, 19
kc_txt, 19
kc_utf8, 19
kto_check, 19
kto_check_blz, 20
kto_check_blz_dbg, 20
kto_check_init, 17
kto_check_pz, 19
kto_check_pz_dbg, 20
kto_check_retval2dos, 19
kto_check_retval2html, 19
kto_check_retval2txt, 19
kto_check_retval2txt_short, 19
kto_check_retval2utf8, 19
kto_check_str, 19

lut_aenderung, 21
lut_bic, 21
lut_cleanup, 18
lut_filialen, 21
lut_free, 18
lut_info, 17
lut_init, 17
lut_loeschung, 21
lut_multiple, 22
lut_nachfolge_blz, 21
lut_name, 21
lut_name_kurz, 21
lut_nr, 21
lut_ort, 21
lut_pan, 21
lut_plz, 21
lut_pz, 21
lut_status, 17
lut_valid, 18

read_lut_block, 17

write_lut_block, 17