Anleitung zum Kode-Konverter KOKO

1. Konverter-Arten

In Verlagen, Setzereien, DTP-Studios und bei Autoren, die mit der Herstellung von Büchern, Zeitschriften, Werbebroschüren usw. befaßt sind, besteht oft Bedarf an Umwandlungs- oder Konvertierungsprogrammen (= Konvertern) für Textdateien. Dabei lassen sich zwei Typen von Konvertern unterscheiden:

Interne Konverter dienen der Umwandlung von der einen internen Norm des Textprogramm A in eine gewünschte andere interne Norm des Textprogramms B, z.B. der Umwandlung von MS-Word-Texten in Word-Perfect-Texte und umgekehrt.

Externe Konverter dienen der Umwandlung von der einen externen Norm A, z.B. der Hausnorm des Verlages A, in eine gewünschte andere externe Norm B, z.B. in die hausinterne Norm der Setzerei B.

Die für die externe Konvertierung, d.h. für den *Datenaustausch* bestimmten Textdateien sind in der Regel *kodierte ASCII-Dateien*, bei denen die Sonderzeichen (Griechisch usw.), die Schriftschnitte (halbfett usw.) sowie die Überschriften, Fußnoten und sonstigen Absatzsorten nach dem Inline-Verfahren kodiert werden. Vom *Inline-Verfahren* (im Gegensatz zum Pointer-Verfahren) spricht man dann, wenn die Kodes *vor der*, *nach der* oder allgemein *in der* jeweiligen Textstelle (»in the line of text«) eingefügt werden, und zwar durch entsprechende Zeichenfolgen wie z.B. #333 für ein bestimmtes Sonderzeichen oder *H1 für eine bestimmte Headline usw. Dabei ist es irrelevant, ob die auf diese Weise kodierten ASCII-Texte später mittels konventioneller Satzprogramme oder mit Hilfe von DTP-Programmen weiterverarbeitet werden. Da sich brancheneinheitliche Kodierungen in der Art der SGML-Norm bei Autoren, Verlagen und Setzereien bislang nicht durchsetzen konnten, müssen häufig kodierte Texte von der einen in die andere Hausnorm umgewandelt werden.

2. Kode-Konverter

Das für IBM-PCs gedachte DOS-Programm **KOKO** (Kode-Konverter) dient der Umkodierung von ASCII-Texten, die für den externen Dateiaustausch nach dem Inline-Verfahren kodiert worden sind. Die Umkodierung erfolgt, indem das Programm KOKO mittels einer frei definierbaren Kodetabelle (Kodedatei) die nach der einen Hausnorm kodierte Textdatei (Altdatei) in die nach einer anderen Hausnorm kodierte Textdatei (Neudatei) umwandelt. Zur Umkodierung der jeweiligen *Altdatei* greift das KOKO-Programm also auf die erforderliche *Kodedatei* zurück und erzeugt eine umkodierte *Neudatei*, indem es die *Altkodes* durch die *Neukodes* ersetzt. Die Kodedatei ist nichts anderes als eine Alt/Neukode-Tabelle oder eine Such/Ersetz-Tabelle.

Suchen und Ersetzen: Die Funktionsweise des KOKO-Programms läßt sich deshalb leicht begreifen, wenn man mit der Search-and-Replace-Funktion (»Suchen und Ersetzen«) eines beliebigen Textprogramms vertraut ist. Wenn man z.B. nur den einen einzigen Kode #333 der Norm A (= Altkode) durch den Kode <<S111>> der Norm B (= Neukode) zu ersetzen hätte, dann würde man keinen Konverter benötigen, weil man diesen einen Kode mit dem Textprogramm selbst bequem austauschen könnte. In der Regel enthalten jedoch kodierte Texte oft Dutzende, bei wissenschaftlichen Werken sogar oft Hunderte von verschiedenen Kodes. Hinzu kommt, daß gelegentlich die ASCII-Norm selbst umkodiert werden muß, z.B. von IBM-ASCII in Mac-ASCII usw. Selbst wenn bei einem Textprogramm die »Find-and-Change«-Funktion nicht nur auf die sichtbaren Schriftzeichen, sondern auch auf die unsichtbaren Steuerzeichen angewendet werden könnte und selbst wenn man sich ein riesiges, zig tausend Bytes langes Befehlsmakro zum automatischen Auswechseln aller Kodes anlegen könnte, wäre die Umkodierung mit dem Textprogramm viel zu langsam, viel zu unflexibel und viel zu fehlerträchtig. Deshalb wurde der nachfolgend beschriebene Kode-Konverter namens KOKO entwickelt, der ASCII-Texte sicher und mühelos sowie zudem extrem schnell umkodieren kann.

3. Programmdiskette

Die Programmdiskette enthält das eigentliche *Konverterprogramm* (KOKO.EXE und KOKOX.EXE) sowie ein konfigurierbares *Stapelprogramm* (KOKOFOR.BAT), diverse Kodetabellen bzw. *Kodedateien* (ASC-256.TAB, BIN-256.TAB, ASC-STAT.TAB, BIN-STAT.TAB, WD-ASC.TAB, WD-VP.TAB, CTRL-DEL.TAB, CTRL-DEZ.TAB), einige Texte und Kodedateien als *Übungs- und Testdateien* (SANS.TAB, SANS.ALT, SANS.ASC; ZEICHEN.DOC, ZEICHEN.TXT; TEST1.TAB, TEST2.TAB, TEST3.TAB).

4. Installation

Zur Installation lege man die Programmdiskette ein und starte die Installation mit **a:install**, wobei ein Verzeichnis **c:\koko** für die Konverterprogramme und die Kodedateien sowie ein Verzeichnis **c:\alt** für die Altdateien (= die umzuwandelnden Texte) und ein Verzeichnis **c:\neu** für die Neudateien (= die umgewandelten Texte) angelegt werden. Die Namen der Verzeichnisse sind jedoch nicht obligatorisch und dienen nur zur Erläuterung der Funktionsweise des KOKO-Programms und für ein kleines Demo-Programm, das nach der Installation automatisch abgespult wird.

5. KOKO und KOKOX

Für die Konvertierung eines bestimmten Textes benötigt man nur den entsprechenden Konverter und die jeweilige Kodedatei. Der Kode-Konverter wird in zwei Versionen geliefert: KOKO.EXE ist die normale Version und KOKOX.EXE ist die erweiterte (extended) Version des Kode-Konverters. Beide Versionen sind funktionsgleich und austauschbar, doch bestehen folgende technische Unterschiede:

Bei der Normal-Version **KOKO.EXE** beträgt die maximale Absatzgröße 8000-12000 Zeichen und die Kodedatei darf neben den 256 1:1-Kodes nur bis zu 300 m:n-Kodes enthalten; dafür ist KOKO erheblich schneller als KOKOX.

Bei der Extended-Version **KOKOX.EXE** beträgt die maximale Absatzgröße 16000-24000 Zeichen und die Kodedatei darf neben den 256 1:1-Kodes bis zu 1300 m:n-Kodes enthalten; dafür ist KOKOX erheblich langsamer als KOKO.

Die jeweilige *Kodedatei* (Kodetabelle) darf bei beiden Programmversionen bis zu 25000 Zeichen umfassen. Der Größe der jeweils umzuwandelnden *Textdatei* (Altdatei) sind bei beiden Versionen keine Grenzen gesetzt.

6. ASCII-Texte

In der EDV wird heute kaum ein Begriff so mehrdeutig verwendet wie das Akronym ASCII (eig. American Standard Code for Information Interchange, d.h. ursprünglich amerikanische Byte-Kodetabelle für den Datenaustausch). Aus der Sicht des Kode-Konverters müssen wir zwischen Zeichen (Bytes), Absätzen (Byte-Folgen) und Texten (Absatz-Folgen) unterscheiden:

Ein **Zeichen** wird bei ASCII-Dateien computerintern durch exakt 1 Byte dargestellt, so daß es 256 verschiedenen 1-Byte-Zeichen geben kann, die man in sichtbare *Druckzeichen* (Buchstaben, Ziffern und Sondernzeichen) sowie in unsichtbare *Steuerzeichen* (z.B. Absatzende, Seitenende, Tabulator usw.) unterteilt. Nach welchen internationalen Normen (ISO-Normen) oder herstellerspezifischen Normen (Macintosh-ASCII, Atari-ASCII usw.) die ASCII-Zeichen dabei intern verschlüsselt sind, ist für den Kode-Konverter völlig belanglos, da das KOKO-Programm jede beliebige ASCII-Tabelle in jede beliebige andere ASCII-Tabelle konvertieren kann, denn selbst der Begriff IBM-ASCII wäre mehrdeutig, weil es verschiedene länderspezifische ASCII-Tabellen gibt, für z.B. Deutsch, Russisch, Arabisch usw.

Unter den ASCII-Zeichen der diversen ASCII-Tabellen nimmt *Ctrl-Null* (NUL, Dezimalkode 0) eine Sonderstellung ein, weil es bei allen mir bekannten ASCII-Tabellen als "Nichts" (NUL) definiert wird und deshalb in normalen ASCII-Dateien nicht vorkommt. Ctrl-Null wird beim Kode-Konverter im ASCII-Modus, d.h. im Nicht-Binär-Modus, programmintern als Datei-Endmarke genutzt.

Ein **Absatz** ist eine Folge von Zeichen, die durch ein Absatzende begrenzt wird. Das *Absatzende* wird bei IBM-ASCII-Texten intern durch die *zwei* Steuerzeichen CR (= Carriage Return, Wangenrücklauf, Dezimalkode 13) und LF (= Linefeed, Zeilenvorschub, Dezimalkode 10) dargestellt. Da die Sequenz CR-LF bei allen IBM-ASCII-Texten zur Darstellung des Absatzendes verwendet wird, ist der für IBM-PCs konzipierte Kode-Konverter darauf eingerichtet.

Ein **Text** ist eine Folge von Absätze. Dabei kann der Kode-Konverter Textdateien beliebiger Größe (von 0 bis x Bytes) konvertieren, also sowohl Kleinstbeiträge für Zeitschriften wie auch Buchtexte mit einem Umfang von vielen Megabytes.

7. ASCII- und Binär-Datei

ASCII-Texte in dem für den ASCII-Kode-Konverter definierten *engeren* Sinne sind Texte, die erstens kein Ctrl-Null enthalten dürfen, bei denen zweitens CR-LF als Absatzende interpretiert wird und bei denen drittens ein einzelner Absatz aus programmtechnischen Gründen eine bestimmte Länge nicht überschreiten darf. Der Kode-Konverter KOKO kann jedoch nicht nur im ASCII-Modus, sondern auch im Binär-Modus betrieben werden. Deshalb müssen wir beim Kode-Konverter zwischen ASCII-Dateien im obigen Sinne und Binär-Dateien unterscheiden, die wie folgt definiert werden:

Eine **ASCII-Datei** ist für den Kode-Konverter eine beliebig große Textdatei, die alle Zeichen von 001 bis 255 außer Ctrl-Null (000) enthalten darf und deren beliebig viele Absätze voneinander durch CR-LF (013-010) abgegrenzt sein müssen, wobei die Absätze jeweils maximal 8000 Bytes (gilt für KOKO) bzw. 16000 Bytes (gilt für KOKOX) umfassen dürfen und infolge der Umkodierung auf maximal 12000 Bytes (gilt für KOKO) bzw. 24000 Bytes (gilt für KOKOX) anwachsen dürfen.

Eine **Binär-Datei** ist für den Kode-Konverter eine mutmaßliche Textdatei, die alle Zeichen einschließlich Ctrl-Null enthalten darf und deren Absätze beliebig lang sein dürfen, wobei CR-LF als Absatzende zu interpretieren versucht wird. Hier liegt es also am Benutzer des Kode-Konverters, ob er dem Programm eine binäre Textdatei oder irrtümlich eine binäre Nicht-Textdatei (z.B. eine Programmdatei oder eine Datenbankdatei) vorgibt.

Der Kode-Konverter kann sowohl ASCII-Textdateien als auch binäre Textdateien konvertieren, doch werden bei Binär-Dateien überlange Riesenabsätze aufgeteilt, so daß Kodes, falls sie sich zufällig an der Teilungsstelle befinden sollten, nicht korrekt umkodiert werden können. In der Praxis spielt diese technische Restriktion jedoch keine Rolle, da z.B. ein 16000 Zeichen langer Riesenabsatz in einem Buch rund 8 Druckseiten umfassen würde, was praktisch nie vorkommt.

8. Programmbenutzung

Die Benutzung des Kode-Konverters setzt keinerlei Fachkenntnisse voraus, *wenn* dem Anwender korrekte Kodedateien an die Hand gegeben werden. Zur Ausführung des Programms KOKO (analog KOKOX) müssen nur drei Dateinamen spezifiziert werden, und zwar die Namen der Kodedatei, der Altdatei und der Neudatei (in dieser Reihenfolge). Die *Kodedatei* enthält die Kodetabelle, die *Altdatei* ist die umzuwandelnde Quelldatei (die alte Textdatei) und die Neudatei ist die zu erzeugende Zieldatei (die neue Textdatei). Beispiel:

KOKO TAB ALT NEU

Obiges Beispiel würde das Programm KOKO.EXE aufrufen, das mittels der Kodedatei (Kodetabelle) namens TAB die alte ASCII-Textdatei namens ALT in die neue ASCII-Textdatei namens NEU umwandeln würde. Man kann aber auch in zwei Schritten vorgehen:

KOKO

Damit wird das Programm KOKO.EXE gestartet, das nun in einer Editierzeile die Eingabe der obigen drei Dateinamen erwarten würde:

TAB ALT NEU

Mit ESC kann man die Editierzeile verlassen. Zum Üben kann man direkt nach der Installation der Programmdiskette folgende Befehlszeile eingeben:

koko sans.tab sans.asc sans.neu /a

(Parameter /a wird weiter unten erklärt).

Das Übungsbeispiel unterstellt, daß sich alle vier Dateien im gleichen Verzeichnis c:\koko befinden, das zugleich das aktuelle Verzeichnis darstellt. Programm-, Kode-, Alt- und Neudatei können sich jedoch in beliebigen Laufwerken und Verzeichnissen befinden. Dann müssen allerdings komplette Pfadnamen angegeben werden, weil sonst jeweils das aktuelle Verzeichnis (Default-Directory) unterstellt wird. Beispiel:

koko sans.tab c:\alt\sans.asc c:\neu\sans.asc /a

Dieses Übungsbeispiel, das ebenfalls direkt nach der Installation ausgeführt werden kann, erwarter KOKO.EXE und SANS.TAB im aktuellen Verzeichnis, während für Alt- und Neudatei komplette Pfadnamen vorgegeben wurden.

Stapelverarbeitung: Wenn man häufig viele kleine Dateien umzuwandeln hat, so kann man im Verzeichnis *c:\alt* die Altdateien ablegen und später im Verzeichnis *c:\neu* auf die dort generierten, gleichnamigen Neudateien zurückgreifen, indem man sich des Stapelprogramms **KOKOFOR.BAT** bedient, das u.a. folgende Befehlszeile enthält, die nachstehend aus Gründen der Übersicht in vier Druckzeilen aufgeteilt wird:

for %%f in (*.*) do
c:\koko**koko**
c:\koko**sans.tab**
c:\alt\%%f c:\neu\%%f /a

Anstelle von *koko* setze man bei bedarf *kokox* ein, und anstelle von *sans.tab* setze man den Namen der erforderlichen Kodedatei ein. Wenn man das Stapelprogramm dann mit *kokofor* (oder genauer mit c:\koko\kokofor) startet, so werden *alle* im Verzeichnis *c:\alt* befindlichen Altdateien mittels der vorgegebenen Kodedatei konvertiert und nacheinander automatisch im Verzeichnis *c:\neu* abgelegt.

9. Parameter

Das Programm KOKO (analog KOKOX sowie analog die oben beschriebene Befehlszeile von KOKOFOR) kann durch eine Reihe von Parametern optimiert und modifiziert werden.

koko /?
zeigt die Liste der Parameter an. Es ist jeweils nur *ein* Parameter zulässig, der nach dem letzten Dateinamen mit einem Schrägstrich eingeleitet wird. Wenn man KOKO *ohne* zusätzlichen Parameter startet, so wird programmintern automatisch der Parameter

/V
simuliert, womit eine ASCII-Konvertierung einegeleitet wird. Das Kürzel »/V« bedeutet, daß eine bereits **V**orhandene Neudatei *nicht* überschrieben wird. Bei *allen anderen* Parametern außer /V wird eine eventuell bereits früher generierte Neudatei *ohne Warnung* überschrieben. Normalerweise sollte man sich der V-Option *oder gar keiner Option* bedienen, weil man dann nie aus Versehen eine Datei überschreiben und damit zerstören kann, z.B. indem man etwa die Kodedatei mit der Neudatei verwechselt usw.

/A
steht ebenfalls für ASCII-Konvertierung, doch wird eine bereits vorhandene, gleichnamige Neudatei ohne Warnung überschrieben.

/Q
steht ebenfalls für ASCII-Konvertierung mit Überschreibung der Neudatei, wobei jedoch weder der Programmablauf noch Warnungen bei der Auswertung der Kodedatei am Bildschirm angezeigt werden (q = quiet), so daß die Konvertierung bei *ausgetesteten* Kodedateien spürbar schneller vonstatten geht.

/S
steht für eine ASCII-Konvertierung mit Überschreibung der Neudatei, wobei jedoch zusätzlich zwei Statistik-Dateien KOKOSTAT.LST (unsortiert) und KOKOSTAT.SRT (sortiert) mit der Häufigkeitsverteilung der m:n-Kodes erzeugt wird. Mittels der auf der Programmdiskette enthaltenen Kodedatei ASC-STAT.TAB, z.B.

koko asc-stat.tab sans.asc sans.neu /s

kann man eine Gesamtstatistik der Häufigkeitsverteilung aller Zeichen (außer Ctrl-Null) der jeweiligen ASCII-Altdatei erzeugen.

/BK
steht für eine Binär-Konvertierung mit Überschreibung der Neudatei. Von diesem Parameter sollte nur im Notfall Gebrauch gemacht werden, z.B. wenn die Altdatei nicht als ASCII-Datei vorliegt oder wenn eine ASCII-Datei exzessiv lange Absätze enthält. Die Binär-Konvertierung ist erheblich langsamer als die entsprechende ASCII-Konvertierung, selbst wenn beide Versionen bei reinen ASCII-Dateien zum selben Ergebnis führen.

/BS
steht für eine Binär-Konvertierung mit Überschreibung der Neudatei, wobei jedoch zusätzlich zwei Statistik-Dateien KOKOSTAT.LST (unsortiert) und KOKOSTAT.SRT (sortiert) mit der Häufigkeitsverteilung der m:n-Kodes erzeugt wird. Mittels der auf der Programmdiskette enthaltenen Kodedatei BIN-STAT.TAB, z.B.

koko asc-stat.tab sans.alt sans.neu /bs

kann man eine Gesamtstatistik der Häufigkeitsverteilung aller Bytes (einschließlich Ctrl-Null) der jeweiligen binären Altdatei erzeugen.

Die Statistik-Optionen (/S und /BS) verlangsamen enorm die Konvertierung, doch sind sie von unschätzbarem Wert bei der Analyse unbekannter Frendtexte und bei der Entwicklung neuer Kodedateien.

10. Kodedatei

Während die weiter oben beschriebene *Programmbenutzung* unter Verwendung bereits ausgetesteter Kodedateien keinerlei Spezialkenntnisse erfordert und in wenigen Minuten erläutert werden kann, weil der Anwender gar nicht wissen muß, welche Umkodierungen vorgenommen werden, setzt die *Entwicklung neuer Kodedateien* umfassende Kenntnisse der Kodierungsstrukturen der Altdateien und der zu generierenden Neudateien voraus, so daß nur Setzer, DTP-Experten und PC-Profis neue Kodedateien anlegen sollten.

Die Kodedatei ist eine ASCII-Datei, die folgenden Aufbau hat:

1. 1:1-Definitionen (256 Def.)
2. Kode-Trenner (z.B. //)
3. Dezimal-Vorkode (z.B. &D)
4. m:n-Definitionen (1-300/1300 Def.)

Kodedateien können mit jedem Textprogramm erstellt und gepflegt werden, das die Abspeicherung von unformatierten ASCII-Dateien zuläßt, was wohl bei allen Textprogrammen möglich ist. Man kann aber auch z.B. den bei DOS 5.0 mitgelieferten Texteditor (EDIT) verwenden.

Man beachte folgende Feinheiten beim Eingeben und Ändern von Kodedateien:

(1) Die 1:1-Tabelle muß aus exakt 256 Zeilen bestehen. (2) Kode-Trenner und Dezimal-Vorkode müssen die 257. und 258. Zeile der Kodedatei bilden. (3) In der sich anschließenden m:n-Tabelle sind aus Gründen der Übersicht und Gliederung reine Leerzeilen (CR-LFs) zulässig. (4) Ein Leerzeichen als letztes Zeichen eines Neukodes muß in der 1:1-Tabelle als 032 und in der m:n-Tabelle als &D032 eingegeben werden, falls der Editor Leerzeichen vor CR-LF nicht darstellt oder beim Abspeichern eliminiert.

Eine Kodedatei kann aus einer *reinen* 1:1-Tabelle *ohne* nachfolgende Definition von Kode-Trenner, Dezimal-Vorkode und m:n-Tabelle bestehen. Dann nimmt das Programm an, daß eine reine Binär-Konvertierung gewünscht wurde, gleichviel ob der Parameter /BK angegeben wurde oder nicht. Wenn jedoch die Kodedatei – wie in der Regel üblich – zusätzlich eine m:n-Tabelle enthält, so müssen nach der 1:1-Tabelle zunächst der Kode-Trenner und der Dezimal-Vorkode definiert werden:

Kode-Trenner: Der Kode-Trenner dient der Definition des Neukodes und steht zwischen Altkode und Neukode. Bei der 1:1-Tabelle ist der Kode-Trenner stets das Gleichheitszeichen, weil hier keine Mehrdeutigkeiten möglich sind. Bei der m:n-Tabelle könnte ebenfalls »=« als Kode-Trenner verwendet werden, falls das Gleichheitszeichen in keinem der Altkodes der m:n-Tabelle vorkommen würde, z.B.
Anton=Berta

Da jedoch die Verwendung eines Einzelzeichens als Kode-Trenner oft zu Mehrdeutigkeiten führt, wähle man als Kode-Trenner eine beliebige, einzigartige Zeichenfolge (z.B. »#*#« oder wie hier »//«), z.B.
Anton//Berta

Dezimal-Vorkode: Bei den Definitionen der Altkodes und Neukodes in m:n-Tabellen können unsichtbare Steuerzeichen vorkommen, die man mit dem Editor nicht direkt eingeben kann. In solchen Fälle ist der 3stellige Dezimalkode einzugeben, der durch einen Dezimal-Vorkode eingeleitet werden muß. Auch hier gilt: Ein Einzelzeichen als Dezimal-Vorkode (z.B. »&«) wäre möglich, wenn dieses Einzelzeichen in den Alt- und Neukodes selbst nicht vorkommen würde. Um jedoch Mehrdeutigkeiten vorzubeugen, wähle man als Dezimal-Vorkode eine beliebige, einzigartige Zeichenfolge (z.B. »*D*« oder wie in den nachfolgenden Beispielen »&D«), z.B.

&D013&D010H1//&D013&D010@H01

(Obiges Beispiel würde alle »H1« am Absatzanfang durch »@H01« ersetzen, weil &D013 für CR und &D010 für LF steht.)

11. 1:1-Tabelle

Die 1:1-Tabelle dient der Umwandlung von einem ASCII-Zeichensatz in einen anderen und besteht aus 256 Definitionen, die folgenden Aufbau haben:

```
000=127
001=001
...
010=010
011=011
012=012
013=013
...
065=A
066=B
067=C
...
```

Links *vor* dem Gleichheitszeichen stehen die Altkodes und rechts *nach* dem Gleichheitszeichen die Neukodes, die mit den Altkodes identisch sein können. Dann erfolgt keine Umkodierung. Die **Altkodes** sind die in der Altdatei, d.h. der umzuwandelnden Datei, verwendeten ASCII-Kodes. Die Altkodes bestehen immer aus der Sequenz 000 .. 255, d.h. sie müssen als 3stellige Zahlen bzw. als Dezimalkodes ausgedrückt werden. Die **Neukodes** sind die ASCII-Kodes, *in die* die Zeichen der Altdatei bei der Generierung der Neudatei umzuwandeln sind. Die Neukodes können wahlweise entweder als 3stellige Dezimalkodes (bei Steuerzeichen erforderlich) oder als 1stellige ASCII-Zeichen (bei Druckzeichen möglich) ausgedrückt werden.

Ctrl-Null: Ctrl-Null (000) darf als *Neukode* niemals rechts vom Gleichheitszeichen eingetragen werden, weil sonst der Kode-Konverter annehmen würde, daß anstelle einer ASCII-Konvertierung eine Binär-Konverierung gewünscht wäre. Da Ctrl-Null in ASCII-Texten niemals vorkommt, kann der Altkode 000 z.B. als Neukode 127 (DEL-Zeichen) umdefiniert werden.

CR-LF: Die Steuerzeichen für das Absatzende, also CR (013) und LF (010), dürfen für die ASCII-Konvertierung niemals umdefiniert werden, weil sonst der Kode-Konverter annehmen würde, daß eine Binär-Konverierung gewünscht wäre, also stets 010=010 und 013=013 für ASCII-Konvertierungen definieren.

Wenn jedoch explizit eine Binär-Konvertierung gewünscht ist (z.B. mit Option /BK), so sind 000, 010 und 013 beliebig umdefinierbar. Derart umkodierte Dateien sind dann aber von normalen Textprogrammen nicht mehr lesbar.

Wenn kein Bedarf an einer 1:1-Umkodierung besteht, weil *dieselbe* ASCII-Tabelle für Neudatei *und* Altdatei gelten soll, so kann man die bereits mitgelieferten Kodedateien ASC-256.TAB (für ASCII-Konvertierung) und BIN-256.TAB (für Binär-Konvertierung) als Ausgangsbasis für die Erstellung neuer Kodedateien mit m:n-Tabellen verwenden. ASC-256.TAB und BIN-256.TAB lassen die ASCII-Kodes völlig unverändert.

Die 1:1-Tabelle wird vom Kode-Konverter immer **simultan** abgearbeitet, d.h. alle 256 Zeichen werden quasi auf einen Schlag umgewandelt. Wenn wir also z.B. definieren

065=B
066=A

so werden alle A-Buchstaben der Altdatei zu B-Buchstaben in der Neudatei und alle B-Buchstaben der Altdatei zu A-Buchstaben in der Neudatei, so daß der weiter unten beschriebene Dreieckstausch entbehrlich ist.

Alle Byte-Byte-Zuordnungen sollten aus Gründen der Performanz nicht in der m:n-Tabelle, sondern in der 1:1-Tabelle erfolgen, weil die *gesamte* 1:1-Tabelle genauso schnell abgearbeitet wird wie *eine* einzelne Definition in der m:n-Tabelle.

12. m:n-Tabelle

Die Definitionen in der m:n-Tabelle haben die Struktur

Altkode//Neukode

wobei nachfolgend »//« als Kode-Trenner und »&D« als Dezimal-Vorkode gelten soll. Beispiele:

#123//<<456>>
ersetzt Altkode #123 durch Neukode <<456>>

{hf}//[bold]
ersetzt {hf} durch [bold]

&D009//[TAB]
ersetzt Tabulator als Steuerzeichen durch sichtbare Zeichenfolge [TAB]

d.h.//das heißt
ersetzt »d.h.« durch »das heißt«

e.g.//z.B.
ersetzt »e.g.« durch »z.B.«

#p//§
ersetzt #p durch Paragraph-Zeichen

&D013//&D013&D010
ersetzt CR durch CR-LF (Binär-Modus)

&D010//&D013&D010
ersetzt LF durch CR-LF (Binär-Modus)

&D032&D032//&D032
ersetzt 2 Leertasten durch 1 Leertaste

&D032&D013&D010//&D032
ersetzt Space-CR-LF durch Space

&D026//
ersetzt EOF-Steuerzeichen durch nichts.

Der *Altkode* (links vor dem Kode-Trenner) kann m = 1 bis x Zeichen und der *Neukode* (rechts nach dem Kode-Trenner) n = 0 bis x Zeichen lang sein, wobei x keine feste Obergrenze hat, d.h. es ist z.B. möglich, anstelle von eigentlichen Satzkodes auch Fachbegriffe, Firmennamen, Überschriften, Abkürzungen usw. durch entsprechende andere Ausdrücke zu ersetzen.

Der Altkode darf niemals leer sein, z.B.
//Berta
denn man kann nichts nicht durch etwas ersetzen. Umgekehrt kann jedoch der Neukode leer sein und führt dann zu einer Eliminierung des Altkodes, z.B.
Anton//
womit alle Antons getilgt werden würden.

Kodes am *Absatzanfang* werden durch »&D013&D010...« und Kodes am *Absatzende* durch »...&D013&D010« eindeutig definiert. Der Kode-Konverter simuliert ein CR-LF vor dem ersten Absatz der Textdatei, so daß auch der allererste Absatz korrekt umkodiert wird.

Doppel-Kodes und Überkreuz-Kodes führen zu einer Warnung, jedoch nicht zum Programmabbruch. Ein **Doppel-Kode** liegt vor, wenn derselben Altkode in der m:n-Tabelle mehrmals auftaucht. Ein **Überkreuz-Kode** liegt vor, wenn ein Altkode auch als Neukode auftaucht. Letzteres ist beim Dreieckstausch unvermeidbar.

Die m:n-Tabelle wird im Gegensatz zur 1:1-Tabelle vom Konverter stets **sukzessiv** und niemals simultan abgearbeitet, so daß ein **Dreieckstausch** erforderlich ist, um zwei Kodes *untereinander* zu vertauschen:

Anton//###
Berta//Anton
###//Berta

Würden wir dagegen *falsch* definieren
Anton//Berta
Berta//Anton
so würde *alle* Namen zu Antons werden.

Die sukzessive Abarbeitung der m:n-Tabelle entspricht der Search-and-Replace-Funktion von Textprogrammen, so daß man leicht überprüfen kann, welche m:n-Umkodierungen bereits bis zu einer bestimmten Stelle der m:n-Tabelle erfolgt sind.

Man beachte im übrigen, daß der Kode-Konverter *zunächst* die 1:1-Tabelle und *erst danach* die m:n-Tabelle abarbeitet, d.h. der Konverter wendet die m:n-Tabelle nicht auf die ursprüngliche Altdatei, sondern auf die gemäß der 1:1-Tabelle möglicherweise bereits erheblich modifizierte Altdatei an.

Das *Löschen* vieler unerwünschter *Einzelzeichen*, z.B. der unnötigen Steuerzeichen im Bereich 000 bis 031, kann effektiver in Kombination zwischen 1:1-Tabelle und m:n-Tabelle wie folgt vorgenommen werden:

000=127
001=127
...
127=127
...
255=255

//
&D
&D127//

Im obigen Beispiel werden alle zu löschenden Einzelzeichen auf das ebenfalls zu löschende DEL-Zeichen (127) umdefiniert und dann mit einer einzigen m:n-Befehlszeile »&D127//« allesamt eliminiert. Mittels der auf der Programmdiskette enthaltenen Kodedatei CTRL-DEL.TAB kann man unnötige Steuerzeichen aus ASCII-Dateien entfernen, und mit der Kodedatei CTRL-DEZ.TAB kann man für Analyse-Zwecke alle Steuerzeichen von Binär-Dateien sichtbar machen.

Bei KOKOX kann die m:n-Tabelle maximal 1300 Definitionen enthalten, wobei die gesamte Kodedatei rund 25000 Bytes groß sein kann. Dies reicht in der Praxis immer aus. Notfalls könnte man aber eine Riesen-m:n-Tabelle in mehrere Kodedateien aufteilen und die Altdatei dann in mehreren Stufen umwandeln, z.B.

Alt zu Temp mittels Tab1
Temp zu Neu mittels Tab2

13. Sonstiges

Windows: Das KOKO-Programm kann unter Windows 3.1 als DOS-Applikation gestartet werden. Eine entsprechende PIF-Datei befindet sind bereits auf der Programmdiskette.

Network: Der Kode-Konverter ist – ähnlich wie ein Textprogramm – bedingt netzwerktauglich. Die *Kodedateien* werden im Read-Shared-Modus geöffnet, so daß man diese Kodetabelllen zur gemeinsamen Nutzung aller User auf dem Server pflegen kann. Da das KOKO-Programm jedoch Hilfsdateien in *dem* Verzeichnis anlegt, *in dem* sich das gestartete Programm befindet, sollte sich eine Kopie des Programms KOKO (analog KOKOX) im Home-Verzeichnis des jeweiligen Users befinden. Das gleiche gilt für die Alt- und Neudateien, die ohnehin nicht im Share-Modus von mehreren Users gleichzeitig bearbeitet werden könnten.

Prozessor: Das Programm KOKO (analog KOKOX) nutzt die erweiterten Befehle der 80386/80486-Prozessoren, falls es diese Prozessoren entdeckt. Dies ist möglich, weil die extrem optimierten Assemblerroutinen des Konverters getrennt für 8086/80286 und 80386/80486 angelegt wurden, so daß selbst im DOS-Modus die 32-Bit-Befehle genutzt werden können. So kommt es, daß die Find-and-Replace-Routinen des Kode-Konverters sage und schreibe bis zu hundertmal schneller sind als die entsprechenden Routinen von Textprogrammen. Mit den Kodedateien TEST1.TAB, TEST2.TAB und TEST3.TAB kann man bei Bedarf in Verbindung mit der Altdatei SANS.ASC entsprechende Zeitmessungen durchführen. So benötigt z.B. das Testprogramm TEST1.BAT auf einem 10-MHz-80286-PC nur 9 Sekunden gegenüber 1000 Sekunden (16 Minuten) mittels der Search-Replace-Funktion des DOS-5.0-Editors EDIT.

I/O-Puffer: Speed-Freaks können die Parameter /A, /Q usw. durch eine Zahl von 1 bis 9 zur Definition des programminternen Absatzpuffers ergänzen. Der Absatzpuffer enthält je nach Puffergröße mehr oder weniger viele Absätze oder im Grenzfall nur einen einzigen Monsterabsatz. Als Default-Wert gilt die als Idealwert ermittelte Puffergröße von 3, insbesondere bei Q3. Theoretisch gilt: Je kleiner die durchschnittliche Länge der Absätze in der Altdatei und je größer die Zahl der Treffer, d.h. der in der Altdatei enthaltenen Altkodes, desto kleiner sollte der Puffer sein (3 bis 1). Umgekehrt: Je größer die durchschnittliche Länge der Absätze in der Altdatei und je kleiner die Zahl der Treffer desto größer sollte der Absatzpuffer sein (4 bis 9).

Debugging: Der Debug-Parameter ist nur für PC-Profis und in keinem Fall für die alltägliche Anwendung gedacht.

Ulrich Stiehl, 1993/2002