Quick Reference Manual for Koko

0. Purpose of KOKO (Kode-Konverter = Code-Converter)

Koko is an extremely fast machine-language search-and-replace DOS program converting a textfile (**oldfile**) to a new textfile (**newfile**) by definable search-and-replace table (**codefile**), consisting of 256 1:1 byte equations and up to 1300 m:n equations.

There is no limit on the size of oldfile. The bigger the textfiles, the more efficient is Koko, compared to wordprocessors, whose search-and-replace function collapses under big files.

1. Two program versions

There are 2 program versions. The faster version is usually sufficient for most applications.

- KOKO.EXE is faster, but codefile is limited to a maximum of 300 m:n equations
- KOKOX.EXE is slower, but codefile can comprise up to 1300 m:n equations

2. Command line syntax

```
koko oldfile newfile codefile /parameter
```

Example:

```
koko sanskrit.txt sanskrit.itx ree-itx.skt
```

would convert **oldfile** sanskrit.txt to **newfile** sanskrit.itx using **codefile** ree-itx.skt

3. Batch processing

The most efficient method of using Koko is by batch processing:

- Create a directory for oldfiles, e.g. **c:\old**
- Create a directory for newfiles, e.g. **c:\new**
- Create a directory for Koko program and for the various Koko codefiles, e.g. **c:\koko**

Create batchfiles such as **k.bat** etc. with following two lines:

```
cd\old
for %%f  in (*.*) do  c:\koko\koko.exe  c:\koko\ree-itx.skt  c:\old\%%f  c:\new\%%f  /q
```

Starting k.bat at DOS prompt would convert **all** oldfiles in c:\old to newfiles in c:\new using codefile ree-itx.skt. For another codefile, e.g. csx-itx.skt, the following change would do:

```
for %%f  in (*.*) do  c:\koko\koko.exe  c:\koko\csx-itx.skt  c:\old\%%f  c:\new\%%f  /q
```

NB: Koko supports only short DOS-filenames (xxxxxxxx.yyy, 8.3 format), not long file names.

4. Parameters

Parameters can be used to control the conversion process. Some of these are the following:

koko oldfile newfile codefile **/v**

converts in ascii mode (default mode) and **does not overwrite** already existing newfile

koko oldfile newfile codefile **/bk**

converts in binary mode (not explained in this quick reference manual)

koko oldfile newfile codefile **/q**

converts in ascii mode quietly (fastest mode) and **does overwrite** already existing newfile

5. Statistics

Koko is supplied with the ready-to-run statistics codefile asc-stat.tab, which is **very useful** for analysing files with undocumented oder incompletely documented encodings.

koko **asc-stat.tab** oldfile newfile **/s**

generates **kokostat.lst** and **kokostat.srt** on the undocumented oldfile revealing what codes are actually used and how often they are used thus often detecting stray codes.

6. Structure of codefile

The codefile is a plain textfile that can be edited with EDIT.COM or any other ascii editor. Warning: Never use Winword, which destroys several codes when re-saving plain txt-files.

The overall structure of codefile is as follows:

1. 1:1 equations (always 256 equations)
2. Definition of m:n separator (e.g. //)
3. Definition of decimal code indicator (e.g. &D)
4. m:n equations (up to 1300 equations)

7. One-to-one equations (1:1)

Koko is supplied with **ASC-256.TAB** used as the **starting point** for creating a new codefile for textfile conversion. Codefile asc-256.tab contains the 256 not-yet-modified 1:1 equations:

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

To the left of "=" always the 3-digit ascii code number must be used. To the right of "=" you can use either 3-digit ascii code (this is obligatory for control codes below ascii 032 = space), or you can use the 1-byte ascii character itself. Some examples:

065=B

066=A

This definition would swap A by B

065=a

066=b

This definition would change A to a and B to b (uppercase/lowercase conversion)

Warning: Koko refuses to work, if 1:1 equations are faulty. There must be always 256 lines of equations with always 3 digits to the left, and always either 3 digits or 1 byte to the right. For instances "065=Aa" oder "065=A " (space after A) would not be tolerated by Koko.

8. Removal of unwanted one-byte-codes

The following fragment shows how unwanted codes can be most efficiently removed:

```
000=127
001=001
...
254=127
255=127
//                    Definition of m:n separator
&D                    Definition of decimal code indicator
&D127//
```

All codes to be removed entirely are redefined as 127, and all unwanted codes marked thus are then removed with this single m:n definition &D127// replacing them all by nothing.

Important: For conversion of ascii files, the first equation must always be 000=127, because code 000 is not allowed in textfiles. Conversion of binary files with 000 is not explained here.

9. Definition of m:n separator and decimal code indicator

In the codefile, after the first 256 lines with 1:1 equations, the lines 257 and 258 are reserved for definition of m:n separator and decimal code indicator. The m:n definitions which follow must be separated by a unique separator, e.g. // or /-/ or ||| or any other unique sequence, and for control codes and special ascii codes, the 3-digit decimal code must be preceded by &D or any other unique sequence indicating that what follows is a 3-digit decimal byte code.

The customary definition is // for separator and &D for decimal code indicator (see above).

10. Simple m:n equations

The application of m:n equations is best illustrated by examples:

```
Sanscrite//Sanskrit
```
would replace Sanscrite by Sanskrt

```
rubbish//
```
would replace rubbish by nothing. **Warning:** Watch out that there is no space after //

```
&D032&D032//&D032
```
would replace two spaces by one space thus removing unwanted double spaces.

```
&D032&D013&D010//&D013&D010
```
would remove space before CR LF (carriage return linefeed)

```
&D013&D010&D013&D010//&D013&D010
```
would replace 2 CR LF by 1 CR LF

11. Complex m:n equations

Some textfiles use CR LF, others use LF only. The following tricky equations

```
&D001//               (This removes byte 001 from oldfile, should it be contained there)
&D013&D010//&D001
&D010&D013//&D001
&D013//&D001
&D010//&D001
&D001//&D013&D010
```

would restore the standard DOS/Windows convention of CR LF (carriage return, linefeed).

Important: In textfiles, paragraphs must be terminated by CR LF or by LF. Otherwise they are non-textfiles. (For non-textfiles, Koko must be used in binary mode with parameter /bk).

The following tricky equations

```
|// |
| |//||
  |// |
||//||&D032
||&D032&D032//||&D032
| &D013&D010//|&D013&D010
|| &D013&D010//|&D013D&D010
```

would standardize dandas at the end of sanskrit lines in a way that there is always one space before first double || and before first single |, and that there is always one space after the first double ||, so that śloka numbers look good, when converted by itranslator.

The following 1:1 definitions are Ulrich Stiehl's own encodings for Sanskrit transliteration:

```
192=ā
193=ī
194=ū
195=ṛ
197=ṝ
198=ḷ
199=ṅ
200=ñ
201=ṇ
202=ṭ
203=ḍ
204=ś
205=ṣ
206=ṃ
207=ḥ
```

Hence the following m:n equations convert Ulrich Stiehl's own transliteration to itx format:

```
&D192//A
&D193//I
&D194//U
&D195//R^i
&D197//R^I
&D198//L^i
&D199//~N
&D200//~n
&D201//N
&D202//T
&D203//D

ch//Ch
c//ch

&D204//sh
&D205//Sh
&D206//M
&D207//H

'//.a
```

The following very complex sequence of equations concatenates Sanskrit ligatures to "_":

```
&D001Rem01//Ligatures

g ai//g_ai
g au//g_au
g a//g_a
g ā//g_ā
g i//g_i
g ī//g_ī
g u//g_u
g ū//g_ū
g ṛ//g_ṛ
g e//g_e
g o//g_o

ṅ ai//ṅ_ai
ṅ au//ṅ_au
ṅ a//ṅ_a
ṅ ā//ṅ_ā
ṅ i//ṅ_i
ṅ ī//ṅ_ī
ṅ u//ṅ_u
ṅ ū//ṅ_ū
ṅ ṛ//ṅ_ṛ
ṅ e//ṅ_e
ṅ o//ṅ_o

ḍ ai//ḍ_ai
ḍ au//ḍ_au
ḍ a//ḍ_a
ḍ ā//ḍ_ā
ḍ i//ḍ_i
ḍ ī//ḍ_ī
ḍ u//ḍ_u
ḍ ū//ḍ_ū
ḍ ṛ//ḍ_ṛ
ḍ e//ḍ_e
ḍ o//ḍ_o

d ai//d_ai
d au//d_au
d a//d_a
d ā//d_ā
d i//d_i
d ī//d_ī
d u//d_u
d ū//d_ū
d ṛ//d_ṛ
d e//d_e
d o//d_o

n ai//n_ai
n au//n_au
n a//n_a
n ā//n_ā
n i//n_i
n ī//n_ī
n u//n_u
n ū//n_ū
n ṛ//n_ṛ
n e//n_e
n o//n_o

b ai//b_ai
b au//b_au
b a//b_a
b ā//b_ā
b i//b_i
b ī//b_ī
b u//b_u
b ū//b_ū
b ṛ//b_ṛ
b e//b_e
b o//b_o

m ai//m_ai
m au//m_au
m a//m_a
m ā//m_ā
m i//m_i
m ī//m_ī
m u//m_u
m ū//m_ū
m ṛ//m_ṛ
m e//m_e
m o//m_o

y ai//y_ai
y au//y_au
y a//y_a
y ā//y_ā
y i//y_i
y ī//y_ī
y u//y_u
y ū//y_ū
y ṛ//y_ṛ
y e//y_e
y o//y_o

r ai//r_ai
r au//r_au
r a//r_a
r ā//r_ā
r i//r_i
r ī//r_ī
r u//r_u
r ū//r_ū
r ṛ//r_ṛ
r e//r_e
r o//r_o

v ai//v_ai
v au//v_au
v a//v_a
v ā//v_ā
v i//v_i
v ī//v_ī
v u//v_u
v ū//v_ū
v ṛ//v_ṛ
v e//v_e
v o//v_o

etc. etc. etc.
_//
```

With the final equation _// the underscore is removed and concatenation of ligatures is effected in transliterated files.

Remarks: For reasons of program speed, Koko does not allow using remarks in codefiles. However it is possible to define dummy equations as remarks, provided they begin with a control code that never occurs in oldfile, e.g. "&D001Remark01//Here follows the remark". To make m:n equations more legible, **one** blank line is allowed between any two equations.

Swapping requires 3 m:n equations using a control code that is never used in oldfile, e.g.

Nandu//**&D001**
Ulrich//Nandu
&D001//Ulrich

Note: In the first 256 one-to-one equations of the codefile, swapping is done by program.

Ulrich Stiehl, 11th of February, 2002