TECkit Tools
A Text Encoding Conversion toolkit

Jonathan Kew
SIL Non-Roman Script Initiative (NRSI)

Abstract

TECkit is a toolkit for encoding conversions. It offers a simple format for describing the mapping between legacy 8-bit encodings and Unicode, and a set of utilities based on such descriptions for converting text between 8-bit and Unicode encodings. TECkit supports Windows, macOS, and Linux platforms.

Introduction

This project was born out of the need to convert data between legacy 8-bit encodings, often developed for a specific field situation, and Unicode. There is a need for documentation of the existing 8-bit encodings, and for specification of how to map them into the Unicode space, and vice versa; then there is also a need for software to actually perform the conversion.

In an ideal world, encoding conversion would usually happen automatically and transparently, for example by being integrated with applications' support for various file formats. But many of the encodings of interest are "private" to a particular language project, and as such will never be supported by mainstream applications. There will probably always be a need for standalone tools that can be used in isolation to modify files as needed, or integrated into customized solutions for end users.

The Text Encoding Conversion Kit (TECkit) provides some pieces that it is hoped may be useful in addressing encoding conversion needs. It includes:

- A simple text file format for describing the mapping between a byte encoding and Unicode;
- A binary file format for mapping tables between byte encodings and Unicode;
- A compiler for building binary mapping tables from mapping descriptions in the TECkit language;
- A compiler for building binary mapping tables from UTR-22 XML descriptions (provided by Martin Hosken);
- The TECkit "engine", a library that applies mappings to text data;
- Utilities based on the TECkit engine that can be used to apply mappings to plain-text and Standard Format files.

The compiled tables and runtime engine are designed to allow very efficient conversion where most codes have simple one-to-one mappings; in cases where there are a large number of multi-character strings, particularly on the input side of the mapping, runtime efficiency will suffer.

Mapping descriptions

On the Unicode Web site there are files that give the mappings between various ISO and vendor-specific encodings and Unicode. For example, here is an extract from the file 8859-1.TXT:

```
0x40   0x0040 #   COMMERCIAL AT
0x41   0x0041 #   LATIN CAPITAL LETTER A
0x42   0x0042 #   LATIN CAPITAL LETTER B
0x43   0x0043 #   LATIN CAPITAL LETTER C
```

These files describe simple one-to-one mappings between the 8-bit encoding and Unicode; as it happens, Unicode was designed such that most established standards have such simple mappings to Unicode. However, many "special character" solutions, especially for minority languages and scripts, require more complex mappings. The relationship between the 8-bit encoding and Unicode may involve one-to-many, many-to-one, or even many-to-many mappings. For example, if the 8-bit encoding uses overstriking diacritics, it may be desirable to map sequences of *base* + *diacritic* to precomposed characters available in Unicode. Or conversely, single 8-bit values may need to be decomposed into sequences of Unicodes.

There are also cases where a further refinement is helpful: contextually-constrained mappings and character classes. For example, in mapping the character *sigma* from SIL Basic Greek to Unicode it may be desirable to map it to either U+03C3 (lowercase *sigma*) or U+03C2 (final lowercase *sigma*), depending whether it is word-final, as Unicode does not expect "smart" rendering of the *sigma*. This could be done by exhaustively listing all combinations of *sigma* with following letters as two-character strings, but it is easier thought of as a single-character mapping constrained by the following context.

Another situation is where character ordering differs significantly between the legacy encoding and Unicode. For example, the byte encoding may encode Devanagari *short i* in its visual position before the consonant (or cluster), while Unicode encodes it in its logical or phonetic position following the consonant (cluster). The mapping description must then ensure that not only is the byte code for *short i* mapped to its Unicode equivalent, but also that it is moved to the proper position in the text stream. This is typically handled by separating the mapping into multiple "passes"; one pass rearranges codes, while another maps between bytes and Unicode characters.

The TECkit mapping file syntax allows these more complex relationships to be expressed in a notation that aims to remain concise and readable while providing considerable expressive power. A separate document, *The TECkit Language*, describes how mapping descriptions are written using this language.

An alternative mapping description language implemented in XML is also available. Unicode Technical Report 22 (UTR-22), available from the Unicode consortium Web site at http://www.unicode.org, describes this format. The current version of UTR-22 does not provide for complex mappings such as those requiring extensive reordering or contextual mappings; proposals to extend the standard to support these features are under discussion at the time of writing.

Mapping table compilers

TECkit includes a Windows command-line compiler, `TECkit_Compile.exe`, that reads encoding files as described above, and generates binary mapping tables for the conversion engine.

The compiler expects the name of the input mapping description file as a command-line argument. The output compiled table file to generate may be specified using the **–o** option; if this is not specified, a default filename (ending with `.tec`) will be used. For example:

```
C:\TECkit\>teckit_compile SILGreek.map -o SILGreek.tec
```

In addition, there is an option **–z** that instructs the compiler to generate the table in an uncompressed format; this is normally only useful when debugging the TECkit compiler or engine. The filename extension '`.tec`' is suggested for compiled TECkit mapping tables, while '`.map`' is suggested for source mapping descriptions.

The `TECkit_Compile.exe` tool also supports a **–u** option that instructs the compiler to read the mapping source text as UTF-8, even without requiring an initial BOM (encoding form signature). Normally, the compiler can auto-detect whether the source text is byte-encoded or Unicode text in any encoding form.

The compiler also supports a **–x** option to generate a XML representation rather than a compiled table. This is primarily intended for use by the Reprise utility, and the XML format produced is subject to change according to the needs of that tool.

The mapping editor described below has been superseded by the EncCnvtrs package, linked to at the end of this document. In addition, the source code for the mapping editor is lost. The old mapping editor binary (from version 2.5.1 in 2006) only ran on Windows, and does not run well on modern versions on Windows (7 and 10). As a result, this binary is no longer distributed. The following description has been retained for reference.

There is also an integrated mapping description editor, compiler, and test environment, `TECkit Mapping Editor.exe`. (This is currently more of a working prototype than a finished product, but is quite usable.) This tool provides a simple text editor that can be used to create and edit mapping description files; these can then be compiled and tested on small text samples using the **Compile** and **Show Test Fields** commands in the **File** menu.

> **Note:** the TECkit Mapping Editor cannot be used to edit mapping descriptions written as Unicode source text; it is strictly an 8-bit text editor. For Unicode mapping descriptions, a Unicode-capable text editor such as Notepad, UltraEdit, etc., should be used, together with the command-line mapping compiler.

Both the command-line compiler and the integrated mapping editor rely on the compiler library, `TECkit_Compiler_x86.dll`, which must be available in the current directory or somewhere in the current Windows PATH. The integrated editor also requires the library `TECkit_x86.dll` (in order to be able to test compiled mapping tables).

The text file conversion tools

One text file conversion tool is included in the TECkit package. This tools can take a compiled mapping, and convert a text file (either from legacy encoding to Unicode or vice versa) using this mapping. The same mapping is applied to the entire file; therefore, this tool are only suitable for simple "text-only" documents where all the content is in a single encoding.

TxtConv: command-line tool

A simple command-line utility, `TxtConv.exe`, is provided that can apply TECkit mappings to plain-text files. There are two required arguments, the input (**–i**) and output (**–o**) files. To apply a mapping description to the text, the compiled mapping table (**–t**) must also be specified (see below for why it might be useful to omit this).

By default, TxtConv applies mappings in the "forward" direction, from the legacy (byte) encoding to Unicode, or from left to right in the mapping description. The **–r** option specifies "reverse" (Unicode to legacy) mapping instead.

Thus, to convert a text file from SIL Basic Greek to Unicode, one might use a command line such as:

```
C:\TECkit\>txtconv -t SILGreek.tec -i greeknt.txt -o greeknt-uni.txt
```

A Unicode application such as Word 2000 should be able to read the resulting Unicode text file.

By default, Unicode output is generated as UTF-8, and the tool attempts to determine the encoding form of Unicode input (when mapping in "reverse") by looking for an initial BOM character. The optional **–of** (output form) and **–if** (input form) arguments allow the user to override this and specify a different encoding form. The recognized forms are:

- `utf8` UTF-8 (default for output, unless input is also Unicode, in which case the default is to use the same encoding form for the output as the input)
- `utf16be` UTF-16 with big-endian byte order
- `utf16le` UTF-16 with little-endian byte order
- `utf16` same as `utf16le` (on Windows)
- `utf32be` UTF-32 with big-endian byte order
- `utf32le` UTF-32 with little-endian byte order
- `utf32` same as `utf32le` (on Windows)

In addition, the form `bytes` may be explicitly given where the input or output is the legacy (byte) encoding. (It is an error to specify a UTF encoding form for legacy text or the byte form for Unicode text.)

When mapping to Unicode, the optional arguments **–nfc** and **–nfd** may be used to request that the Unicode output be normalized to NFC or NFD respectively. (See *Unicode Standard Annex #15: Unicode Normalization Forms* for information on these.) Finally, the argument **–nobom** may be specified to tell TxtConv *not* to include a Byte Order Mark in the output file; otherwise, one will be generated.

TxtConv, as well as the other conversion tools mentioned below, relies on the mapping engine `TECkit_x86.dll`, which must be available in the current directory or somewhere in the current Windows PATH.

Note that the TxtConv tool applies a single mapping table to a complete text file. It is thus not suitable for files that contain data in a mixture of legacy encodings. Such mixed-encoding data requires software that can interpret markup or otherwise determine the encoding of each portion. Such a tool can then use the TECkit mapping engine to apply the appropriate mapping table to each field, text run, or other fragment of data.

If the **–t** (mapping table) argument is not specified, TxtConv will map from Unicode to Unicode using a "null" mapping. While this may seem pointless, it can in fact be very useful: as the input and output encoding forms can be independently specified, this provides a means to transform Unicode text between the five supported encoding forms. In addition, if the **–nfc** or **–nfd** argument is specified, TxtConv will perform normalization of the Unicode text.

An argument to TxtConv is available to control the handling of unmappable input. When the conversion engine encounters data that cannot be mapped between bytes and Unicode using any of the mappings in the table, the normal behavior is to replace the unmapped character with the "default mapping" specified by the table (typically U+FFFD REPLACEMENT CHARACTER when mapping to Unicode, and 0x3F '?' when mapping to bytes). If the optional argument **–u 1** is specified, the tool will print a warning to the terminal if the default is used. And if **–u 2** is specified, the tool will abort processing when an unmappable character is found, and print an indication of where in the input it occurred. (The default behavior of silently using the replacement character can be explicitly requested with **–u 0**.)

DropTEC: drag-and-drop text file converter

The functionality of this tool has *not* been superseded by the EncCnvtrs package. The rest of the comments about the source code, executable binary, and description in this document pertaining to the mapping editor apply equally to DropTEC.

In addition to the command-line tool, TxtConv, there is a text file conversion utility with a simple Windows user interface, DropTEC. This allows TECkit mappings to be applied to plain text files without requiring the use of the Windows command prompt.

DropTEC displays a single window, with "file boxes" for the mapping table, legacy, and Unicode files. First, a mapping table must be loaded by dropping the compiled (`.tec`) file onto the Mapping Table box; then either legacy (byte) or Unicode text files may be dropped onto their respective boxes. DropTEC will prompt for an output file to write the converted text, with the default being to append "`.txt`" to the input file name.

There are Browse buttons and commands in the File menu that provide alternatives to dropping files from Windows Explorer into the file boxes.

When converting to Unicode, the radio buttons in the main window determine which encoding form DropTEC should generate. When converting from Unicode back to the legacy encoding, DropTEC looks for a Byte Order Mark (BOM) at the beginning of the input file to determine the encoding form; if none is found, it will treat the input text as UTF-8.

Converting formatted or marked-up data

The core function provided by the TECkit engine is to convert a contiguous "chunk" of text data, using a single mapping, between two encodings (typically a legacy encoding and Unicode). The text file tools mentioned above apply this operation to complete files; thus, they assume a single encoding for an entire file.

Many documents, however, contain data in a variety of languages, possibly using a variety of encodings, and mix actual text data with markup or formatting information. Examples include multilingual Shoebox databases, where different fields may use different encodings, or Word documents where parts of the text use "custom" fonts with non-standard encodings. It is inappropriate to apply a single TECkit mapping to such documents, as the different sections of the data require quite different mappings. What is needed, therefore, is a tool that can interpret the document structure (whether represented by in-line text markup such as Standard Format markers, or some other scheme such as Word's formatting information), identify the segments of text data, and then apply the proper mapping to each segment.

Converting Standard Format files: SFconv

A tool for converting Standard Format (SF) files is included in the current TECkit package. This is `SFconv.exe`, a command-line tool that uses an XML "control file" to specify the mappings associated with the various markers in an SF file. Note that the current SFconv tool should be considered a prototype rather than a full solution to the issue of SF conversions. A more complete tool with an improved user interface would be desirable.

An annotated example of an SFconv control file should serve to illustrate its use. This is GNT-map.xml, found in the Samples directory of the TECkit distribution:

```
<?xml version="1.0"?>
<!--Copyright (c) 2002 SIL International-->
```

> The control file is an XML file, and as such begins with a standard XML header line. Being an XML file, it is a Unicode document; however, there is normally no need for any characters outside the ASCII character set, in which case a plain ASCII file (which is also valid as a UTF-8 Unicode file) created with a standard text editor will suffice.

```
<sfConversion defaultMapping="SILGreek">
```

The top-level element of the control file is `sfConversion`. This has a single attribute, `defaultMapping`, which gives the name of the TECkit mapping file to be used as a default for any data that does not have any other mapping specified. In this case, SFconv will use the mapping file `SILGreek.tec` for any "unknown" fields in the SF file. It has a subelement `sfMarkers`, and optionally also `inlineMarkers`.

```
<sfMarkers escape="\"
       chars="abcdefghijklmnopqrstuvwxyz_ABCDEFGHIJKLMNOPQRSTUVWXYZ"
       mapping="ISO-8859-1">
```

The `sfMarkers` element defines the Standard Format markers that SFconv should search for in the text. The exact form of SF markers can be specified by use of the (optional) `escape` and `chars` attributes of `sfMarkers`; these give the escape character that introduces markers (default: backslash) and the set of characters that make up marker names (default: A–Z, a–z, 0–9, and underscore). The `mapping` attribute gives the name of the mapping file to be used to convert the markers themselves between the legacy encoding and Unicode (default: the `defaultMapping` of `sfConversion`).

```
    <marker name="id" mapping="ISO-8859-1"/>
    <marker name="fe" mapping="ISO-8859-1"/>
```

The `marker` subelements of `sfMarkers` specify the mappings to be used for the data following specific format markers. Note that data following any markers not explicitly listed here will be mapped using the `defaultMapping` of `sfConversion`.

```
</sfMarkers>
```

In addition to "standard format" markers, SFconv supports "inline" markers that consist of an escape character (distinct from the SF marker escape), a marker name, and "begin" and "end" delimiters for the data to be marked. Markers such as this are sometimes used to tag runs of text within fields of a Shoebox database, for example.

```
<inlineMarkers escape="|" start="{" end="}"
       chars="abcdefghijklmnopqrstuvwxyz_ABCDEFGHIJKLMNOPQRSTUVWXYZ"
       mapping="ISO-8859-1">
```

The `inlineMarkers` element is similar to `sfMarkers`, but can have additional `start` and `end` attributes to specify the delimiters around the tagged data. The defaults are vertical bar as the escape character, and curly braces as start and end delimiters.

```
    <marker name="rm" mapping="ISO-8859-1"/>
    <marker name="gk" mapping="SILGreek"/>
```

Each inline marker specifies the mapping for the tagged data. Note that (unlike with SF markers) this mapping applies only until the ending delimiter of the inline-tagged data, after which the mapping of the prevailing SF marker applies again. Also unlike SF markers (which have no ending delimiter), inline markers may be nested in the data to be converted.

```
    </inlineMarkers>
</sfConversion>
```

An SFconv control file such as this can be used to control conversion of a legacy Standard Format file to Unicode and vice versa.

The SFconv utility uses command-line options to specify the conversion direction, control file, and input and output files. The required arguments are:

-8u | **-u8**	conversion direction: 8-bit to Unicode or vice versa
-c *controlFile*	specifies XML control file
-i *inFile*	specifies filename of input SF file
-o *outFile*	specifies filename for converted output

In addition, the following optional arguments may be used if appropriate:

-d *mappingDir*	directory where mapping files (`.tec` files) are to be found
-utf8 | **-be** | **-le**	Unicode encoding form: UTF-8, UTF-16BE, or UTF16-LE (default is UTF8 for output, or detected from input file if BOM is present)

-bom	(only when mapping to Unicode) write initial BOM to the output file
-nfc | **-nfd**	(only when mapping to Unicode) normalize to NFC or NFD

Converting Microsoft Word documents: VBA macros, RTF

One approach that can be used with Word documents relies on Word's Visual Basic for Applications (VBA) language. VBA macros can search for runs of text in a particular font or style, and then call a TECkit converter to apply a particular mapping to that text. The converted text is then inserted into the document and a new font applied.

Example VBA code showing techniques for calling TECkit from within Word can be found in the Developers directory of the TECkit distribution. This sample code is available to assist those who may wish to use VBA to process Word documents, but does not consitute a polished end-user tool.

For a more complete package for performing encoding conversions within Word documents, including a "repository" for mapping tables, and access to other types of mapping as well as TECkit tables, see the EncCnvtrs package, available separately.[1]

Another option for applying conversions to Word documents is to save the document as RTF, and then use text processing tools to read the RTF, detecting font changes and thus choosing appropriate mappings to apply. However, the complexity of RTF (and changes between the RTF generated by different versions of Word) can make it difficult to do this in a completely general way.

[1] http://scripts.sil.org/EncCnvtrs