RTF Tools
Character Mapping

Paul DuBois
dubois@primate.wisc.edu

Wisconsin Regional Primate Research Center
Revision date: 5 April 1994

1. Introduction

Text characters in an RTF file may be specified as literal characters or using \'*xx* notation, where *xx* is the hex value of the character. RTF files also contain a control word that specifies the character set that's used within the document and governs the interpretation of character values. The charset control words are:

\ansi	ANSI (default)
\mac	Apple Macintosh
\pc	IBM PC
\pca	IBM PC page 850, used by IBM Personal System/2

Although the four charsets don't appear to differ for characters in the ASCII range (below 128), they differ considerably above the ASCII range (128−255). For example, the ANSI, Macintosh, and PC charsets represent the degree sign ("°") as \'b0, \'a1, and \'f8. Furthermore, even for a given charset, character values in the Symbol font represent different characters than they do generally. For example "a" in Symbol font is "α".

The RTF Tools distribution uses a translation model that divides translation effort between reader code that tokenizes the RTF input stream and writer code that decides what to write based on the current token. The reader is common to all translators; writer code differs on a per-translator basis. In order to free writer code from charset dependencies, the reader performs input character mapping onto a charset-independent set of character codes. This makes it unnecessary to duplicate charset-dependent decisions in each writer.

2. Character Translation

Input RTF characters are mapped to output sequences like this:

- When a text character (i.e., a token in the *rtfText* class) is read, the reader maps it onto a "standard" character code, using a translation table selected by taking into account the charset used by the input file and whether or not the font is Symbol. *rtfMajor* and *rtfMinor* are set to the input character value and the standard character code, respectively. If there is no entry for the RTF character in the translation table, *rtfMinor* is set to *rtfSC_nothing* (0).
- At output time, the writer maps the standard character code to an appropriate output sequence. Typically the writer does this by consulting an output map that specifies the appropriate output sequence for each standard character code.

Using this method, the writer has access to the original input character if it needs it for some reason (*rtfMajor*), but also can work with a charset-independent representaton of the character (*rtfMinor*).

2.1. Standard Character Representation

Charset maps and output maps are represented as simple text files that are read when a translator runs. In map files, the standard characters are represented by names patterned after those used in Adobe Font Metric (AFM) files. For example, ‘‘space’’, ‘‘quotedblleft’’, and ‘‘bullet’’ represent the space, left double quote, and bullet characters. When map files are read in, the standard names are converted to integers in the range 0..*rtfSC_MaxChar*−1, where *rtfSC_MaxChar* is the number of standard characters recognized.

The set of standard characters contains characters represented by the charsets. It also contains characters represented by certain RTF ‘‘special-character’’ control words, such as ‘‘nobrkspace’’ for the unbreakable-space control word ‘‘\~’’.

By keeping charset and output sequence maps in files instead of compiling them in, translations can be changed or fixed by editing text files rather than by recompiling programs.

2.2. Map Locations

The charset map files are found within the distribution in the *LIBSRC* directory and are installed in *LIBDIR*. Individual translators provide output maps in their own source directory and install the maps in *LIBDIR*.

The map reading functions *RTFReadCharsetMap()* and *RTFReadOutputMap()* look for charset and output map files in the current directory before looking in *LIBDIR*. This allows users to copy maps from *LIBDIR* into their current directory, modify the copies, and have the translators use the modified copies instead.

2.3. Charset Map Format

General and symbol charset maps are stored in the text files *ansi-gen*, *ansi-sym*, *mac-gen*, *mac-sym*, *pc-gen*, *pc-sym*, *pca-gen*, and *pca-sym*. Each line of a charset file associates an RTF character value (field 2) with the standard character name to which the RTF character corresponds (field 1). Here’s a sample from *ansi-gen*:

```
parenleft       (
parenright      )
space           " "
quotedbl        ´"´
quoteright      "´"
quoteleft       "`"
a               a
b               b
c               c
bullet          0x95
emdash          0x96
endash          0x97
```

Character values may be given as a single character (in which case the ASCII value is used), or as a hex number 0x*yy*. Single or double quotes may be used to quote values containing whitespace or quotes (e.g., use single quotes to quote a double-quote).

Lines with a ‘‘#’’ in column one are taken as comments. Comments and blank lines are ignored.

2.4. Output Sequence Map Format

Writers can encode the map associating standard character codes and output sequences any way they want. However, a map can be stored in a text file and read easily with *RTFReadOutputMap()* if the map file uses a format such that: (i) field 1 is the standard character name, just as in the charset maps; (ii) field 2 is the output sequence to produce for the character named in field 1; (iii) comments are specified as in charset maps.

Here’s part of the *groff-map* file used by *rtf2troff* to produce output for *groff*:

```
a               a
b               b
c               c
quotedblleft    "``"
quotedblright   "''"
bullet          \(bu
emdash          \(em
AE              \(AE
ae              \(ae
```

To read an output file, declare an array of character pointers to hold the output sequences and call *RTFReadOutputMap()*:

```
char   *outMap[rtfSC_MaxChar];

if (RTFReadOutputMap (outputMapName, outMap, 1) == 0)
        RTFPanic ("Cannot read output map %sn", outputMapName);
```

Like *RTFReadCharSet()*, *RTFReadOutputMap()* looks first in the current directory and then *LIBDIR* for the file (unless the file is specified as an absolute pathname).

The last argument to *RTFReadOutputMap()* is 1 or 0, depending on whether or not you want the map to be initialized before the file is read. (It is sometimes useful to construct a map from several sources, in which case you initialize the map only for the first file read, not the second and subsequent ones. See *rtf2troff* for an example.)

To convert a standard character code to an output sequence, use the code as in index into the output map array:

```
seq = outMap[rtfMinor];
```

Output mapping varies from simple to relatively complex. *rtf2text* uses a single output map, *text-map*. For *rtf2troff*, it's not so simple. *rtf2troff* selects an output map that's specific to the version of *troff* you want output for (*xroff-map*, *groff-map*, etc.). This is because special characters are typically represented in *troff* by escape sequences (such as ``\(de'' for the degree symbol), but different versions of *troff* support differing sets of escape sequences. If *rtf2troff* can't find a map for your version of *troff*, it reads a ``generic'' *troff* map.

If an −*m* argument is given on the command line, *rtf2troff* also looks for an output map corresponding to the named macro package and combines it with the *troff*-version map. This is because macro packages sometimes provide escape sequences of their own to support additional special characters.

3. Dealing with Missing Map Entries

Problem 1: The charset maps provided in the distribution are fairly complete. However, it is still possible that a map will have no entry for a given input character. In this case, the reader sets *rtfMinor* to *rtfSC_nothing*. This is a legal character code, and you can treat it various ways. One approach is to put an entry for the name ``nothing'' in your output map, e.g.,

```
nothing      "[[CHARACTER DROPPED]]"
```

This way, when the writer translates characters to output sequences, it need make no special effort for characters that are missing from the charset map, and an obvious marker appears in the output. However, this approach doesn't provide any clue about what the character was.

Alternatively, you can try to show some representation of missing characters in the output:

```
if (rtfMinor != rtfSC_nothing)
        /* print normal output sequence here */
else if (rtfMajor < 128)   /* in ASCII range */
        printf ("[[%c]]", rtfMajor);
else                       /* non-ASCII, print hex value */
        printf ("[[\\´%02x]]", rtfMajor)
```

Problem 2: If there is no entry for a standard character code in the output map, the output sequence will be NULL. One way to deal with this is to print the standard character name:

```
char *seq;

seq = outMap[rtfMinor];
if (seq != (char *) NULL)
        fputs (seq, stdout);
else
        printf ("[[%s]]", RTFStdCharName (rtfMinor));
```

4. Additional Charsets

The mapping mechanism described here handles the four existing charsets uniformly. If/when the RTF specification is modified to allow additional charsets, it should be easy to provide support for them with no change to writer code:

- Add a #define for the charset control word to *rtf.h*.
- Add an entry for the control word to the *rtfKey*[ ] array in *reader.c*.
- Create and install general and symbol charset map files.

With those changes made, translators can simply be recompiled and should automatically work for RTF files using the new charset.