香川大学経済論叢
第78巻 第4号 2006年3月　1－17

日本語・満州語の辞書作成のための補助システム（Ⅳ）

本　田　道　夫

Ⅰ．はじめに

日本語・満州語の辞書作成のための補助システムの開発は，本システムを用いて実際に辞書データの入力をしている方からの要請，および使用環境改善のために筆者自身が気がついたことなどの点で，その都度改良を重ねてきた。主な改良作業は，編集サブシステムについてのものであるが，[本田 [2003]] 執筆時の編集サブシステムのバージョンは1.12であったが，現段階では1.50であり，エラー修正も含めてではあるが，少なくとも38回以上の改良をしていることになる。

補助システムの開発は，おそらく辞書作成の終了まで続くと思われ，まだ中間段階ではあるが，前回から，現時点までの辞書データの入力に用いる編集サブシステムにおける改良について記す。主な改良は以下のものである。

(a)　キーボードからの入力を待つ方式が，ループしながらキー入力の有無を確認する処理方式であったが，その方法では，キー入力がない場合にも無駄にCPUに負荷をかけていたので，キー入力がない場合はCPUに負荷をかけないように改良した。

(b)　シフトJISの文字体系には辞書作成に必要な漢字が揃っていないために，約12万文字を備えている文字鏡システムを利用させていただき，「シフトJIS文字＋文字鏡文字＋満州文字」の文字コード体系を利用することについては［本田2003］に記したが，その利用の詳細は未定であった。

その後，どのように文字鏡文字を内部で取り扱うか，およびどのように

画面に表示するかなどの詳細を確定し，実現した。

(c) 中国に作成を依頼した辞書データは，かなり多くのUnicode文字を用いていた。これを文字鏡文字に置き換えるには，対応する文字鏡文字を一覧から探し，それに本システムのための文字コードを割り当てる必要がある。しかし，文字数が多くしかも今後も同様なUnicodeを用いたファイルでデータを受け取ることが予想され，そのたびに同様な作業が必要となる。また，約3,000文字の追加漢字ということで用意していた本システムでの追加漢字の文字コード割り当て領域がなくなる可能性もでてきた。したがって，Unicode文字については，Unicode文字のまま扱うこととした。つまり，「シフトJIS文字＋文字鏡文字＋満州文字」に加えて，「Unicode文字＋文字鏡文字＋満州文字」も扱えるようにした。

(d) これまでは，編集システムの内部での文字コードはシフトJISとしていたが，(c)で述べたように，Unicodeを扱う関係から，Unicode（UTF 16-BE）とした。

以下，これらの改良について説明する。なお，編集サブシステムのほとんどの部分は，エディタプログラムJMacsであるので，以降では，編集サブシステムJMacs，エディタJMacs，あるいは簡単にJMacsということがある。

Ⅱ．キーボードからの入力を待つMain Loopの処理方式の改良

2.1. これまでのキーボード入力処理の問題

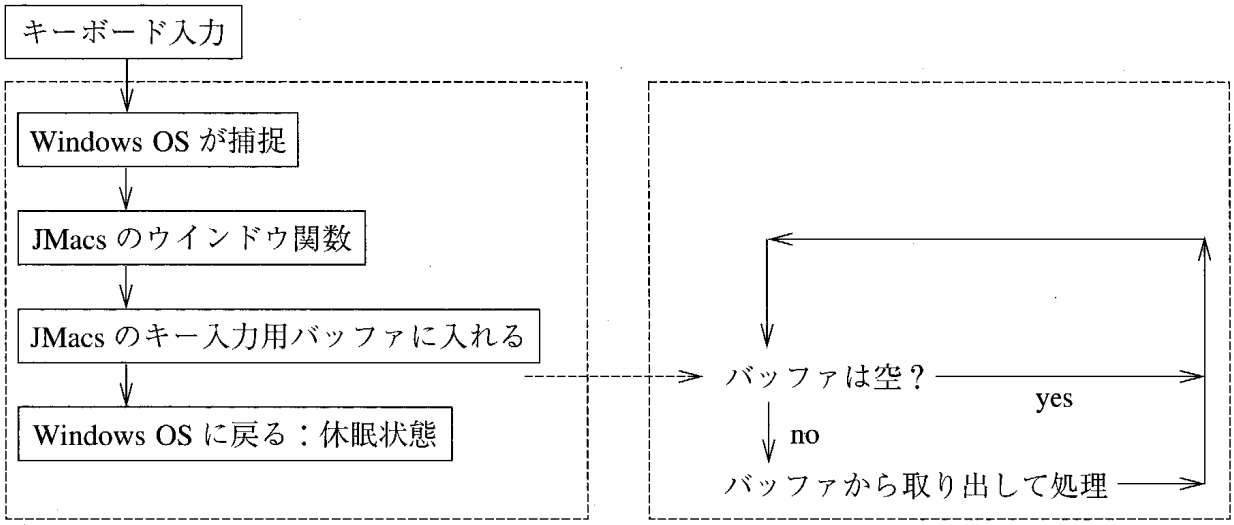
エディタJMacsは，ユーザによるマウス操作あるいはキーボード入力など（以後，これらを，「ウインドウ上での操作」という）を受けて，それに応じた処理をする。マウス操作は，MS-DOSのときにはなかった操作であり，Windowsのプログラムで用いるWin 32 APIの解説書の方法を参考にプログラムを作成したが，キーボード入力に対しては，MS-DOSのときに採用していた従来の方法と同じ方法での処理としていた。まず，従来の処理方式での無駄なCPU負荷の問題を説明する。

マイクロソフトのWindowsオペレーティングシステム（以後，「OS」という）は，複数のウインドウを開き，並行して処理を進めることが可能なマルチプログラム処理方式である。そのために，Widnows OSの場合，各ウインドウに対応するプログラムは，ユーザからの操作を無駄なループ実行により待っているのではなく，操作が行われるまではCPUに負荷をかけない状態（以下では「休眠状態」ということもある）で待っている。つまり，マウス，あるいはキーボードの操作待ちのような状態ではCPUに負荷をかけずに，他のウインドウでの処理にCPUを有効に利用できるようになっている。そして，自分のウインドウ上で操作があって，はじめて操作に応じた処理をするようになっている。

この仕組みは，Win 32 APIを利用するプログラムの場合次のようになっている。各ウインドウ上での操作は，まず，Window OSで捕らえられたのち，Windows OSが，操作の行われたウインドウのウインドウ関数に渡す。各プログラムは自分のウインドウ上での操作がWinodws OSからウインドウ関数に渡されるまでは，CPUに負荷をかけない休眠状態で待機しており，渡されてから処理を行う。つまり，ウインドウ関数に渡されるまでは，プログラムは眠っていて，キー入力が渡されることにより目覚めて処理を開始し，処理終了後はWindows OSに制御を返すことにより，CPUに負荷をかけない休眠状態になるという仕組みである。なお，「ウインドウ関数」は，プログラム中でユーザが定義するものである。

これまでのエディタJMacsでは「図１　これまでのウインドウ上の操作の処理」のように，キーボード入力をプログラム中のキーボード入力を蓄えるバッファに格納するウインドウ関数での処理の流れと，そのバッファを常に監視し，入力があれば処理を行う流れ（図１のキーボード入力の処理の右側のループ部分）の２つがあった。前者の流れは，キーボード入力により，Windows OSから目覚めさせられたウインドウ関数は，渡された入力をバッファに格納して，再度CPU負荷をかけない休眠状態となるが，後者の流れは，常時CPUに負荷をかける流れであった。

図1　これまでのキーボード入力の処理



・キー入力があったときのみ実行　　・常時ループ部分実行（CPU に負荷）

エディタ JMacs は，満州文字や文字鏡文字の機能を利用せずに，通常のエディタとしても利用できるので，開発後にはいろいろな作業のために用いていたが，その際も，あまり他のプログラムと CPU 利用で競合するような事態は生じてなかったために，しばらくの間は，無駄な CPU 負荷については気がついていなかった。しかし，あるとき Windows タスクマネージャを起動し，パフォーマンスを表示させたときに，JMacs だけが動いており，それも入力待ちの状態であるにも拘らず，CPU 利用率が高くなっており，無駄な CPU 負荷に気がついた。このことは，たとえば，JMacs とブラウザを起動していれば，ブラウザの方で作業していても，JMacs は無駄な CPU 負荷をかけ，ブラウザの動作が遅くなることを意味し，マルチタスク処理の Windows OS 上でのプログラムとしては問題である。

2.2. 新たなキーボード入力の処理方式

マウス操作の処理と同じように，ウインドウ関数にキーボード入力が渡されたときに適切な処理をし，終了後はウインドウに制御を返し，休眠状態になるようにすれば，無駄な入力待ちのループは不要となる。しかし，JMacs のキーボードからのコマンドには，「１文字右にカーソルを移す」ような「コントロ

ールF」の１入力だけのコマンドもあるが，インクリメンタル検索のコマンドのように，２つ以上の入力からなり，しかも１入力ごとに処理を進めるものもある。つまり，最初の検索コマンド「コントロールS」の入力でインクリメンタル検索処理のプログラム部分に入り，以後はこのプログラム部分で，ESCあるいは他のコマンドとなるキーボード入力があるまで，キーボード入力を受け取り，検索対象文字列に付け加えながら検索を進める処理である。

そこで，改良は次のような方法とした（図２　改良したキーボード入力の処理）。

(A)　キーボード入力のコマンド処理については，そのコマンド処理中であるか否かを示すフラグ変数を用意する。初期値は，「コマンド処理中ではない」とする。

(B)　Windows OSを通じてのキーボード入力を受け取ったウインドウ関数は，以下のように処理する。

(1)　キー入力を蓄えるプログラム中のバッファに入れる。

(2)　フラグ変数の値が「コマンド処理中」であれば，制御をOSに戻し，休眠状態となる。なお，バッファに入れた入力は，コマンドの２番目以降の入力として取り出され，処理される。

(3)　フラグ変数の値が「コマンド処理中」でなければ，コマンドを実行処理する。

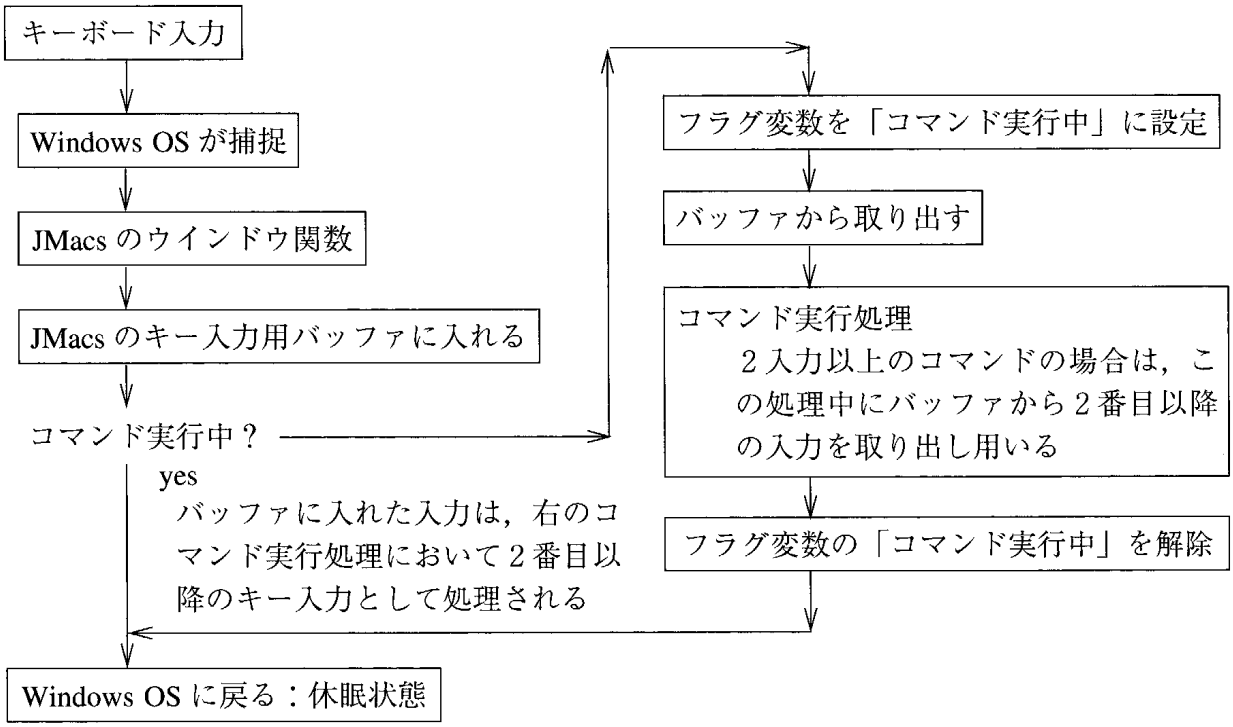
(a)　テキストへの文字挿入（これもそれ自身を挿入するコマンドとして扱う）の場合は，挿入処理を行う。１キー入力のコマンドであるので，これで，コマンド処理は終了し，制御をWindowsに戻し，休眠状態となる。

(b)　それ以外のコマンドの場合は，フラグ変数の値を「コマンド処理中」とし，実行処理を行う。処理終了後にフラグ変数の値を「コマンド処理中」ではないとした後，コントロールをWindowsに戻し，休眠状態となる。

(C)　上記の(3)の(b)で２入力以上のコマンドの場合は，２番目以降の入力はそ

のコマンド処理中で，(1)でバッファに入力されるまで待ち，入力があれば，それを用いて処理を続ける。

図2　改良したキーボード入力の処理



新しい処理方法でも，上記(C)に記した，２入力以上のコマンドについては，それぞれのコマンド処理の部分で，ウインドウ関数によりバッファに入れられる入力を待つループとしている。しかし，そのループの実行は，一連のキーボード入力の２入力目以降を待つ場合のみであり，以前のように，JMacs を起動しているがコマンド処理をしていないという状態のときのような無駄な CPU 負荷をかけることはない。

Ⅲ．文字鏡文字の扱い

3.1. 文字鏡文字の利用

このシステムを開発する当初は，約3,000文字程度の追加漢字を扱う必要があるということであり，最初に実現した日本電気のパソコン PC-9801 上では，満州文字と同様に，画面用と印刷用の漢字をそれぞれドットフォントとして作成するためのシステム開発，およびそれを利用してのフォント作成を行った。

システムを Windows OS 上に移すときに満州文字（200 文字弱）はドットフォントを True Type フォントに変換したが，その作業が大変であった。また，漢字の数は多く，しかも画数が多いため，同じ変換では綺麗な文字とならない可能性もあり，漢字フォントを探していたところ，文字鏡文字というフォントシステムの存在を知った。字数も 12 万文字と多く，フォントも綺麗で，また文字の出典が確かなものであれば，作成してもらえるとのことであったので，すでに，［本田 2003］で述べたように，文字鏡を利用することにした。

3. 2. 文字鏡文字の文字コード

文字鏡の各文字には番号が割り当てられているが，それはシフト JIS のコードと重複している。したがって，文字鏡の文字番号をそのまま本システムで用いることはできない。

文字鏡の表示方法（利用方法）として文字鏡研究会の Web ページ（http : // www.mojikyo.org/html/index.html）で説明されているものは，文字鏡文字の番号ではなく，フォントセット名とそのフォント内での 16 ビットのコードの組を用いるというものである。このフォント名と文字コードは，同 Web ページで提供される「Mojikyo Character Map」というソフトウエアを用いて，文字鏡番号から得ることができる。しかし，フォント名と文字コードという方法は，本システムでの文字コードの扱い，すなわち，シフト JIS で用いていないユーザ定義領域のコード（16 ビット）を，満州文字と追加漢字に割り当てるという方法とは合わない。

そこで，従来からの追加漢字と同様に，シフト JIS のコード体系のユーザ文字定義用に用意されているコード領域のうち 16 進数表現で F000 以降で，満州文字に割り当てている部分以外の部分に割り当てることにした。具体的には，F000～F9FF，FC00～FFFD としている（現時点では，F000～F9FF のみ利用）。また，文字鏡の文字すべてを扱うのではなく，必要な文字についてのみ上記のコード領域に追加的に割り当てることとした。

3.3. 文字鏡文字の表示

文字鏡文字の表示は,文字鏡番号ではなく,その番号に対応するフォントセット名とそのフォント内でのコードで行う，すなわち，フォントセット名のものにフォントを切り替えたのち，そのフォント内での文字コードを出力すればよい。なお，このことは，文字鏡文字を用いるという決定をした時点では，確認はしていたが，どのようにシステムに取り込むかは確定していなかった。

当初は，次のような手順を考えていた。

(1) 本システムを用いて辞書を作成する先生が，データ入力していく途中で，追加漢字が必要となったときに，該当する文字を文字鏡システムを利用して見つける。

(2) その先生，あるいは本田が，本システムでの文字コードと文字鏡での文字の番号の対応を決める。

(3) 文字鏡での文字番号に対して，「Mojikyo Character Map」で，追加漢字に対するフォント名と文字コードを手作業で調べて，

本システムでの文字コード，フォント名，文字コード

のような情報の配列とし，プログラムに埋め込む。

しかし，その時点で既に確定していた追加漢字の数，および今後さらに増えるであろうことを考えると，フォント名と文字コードを手作業で調べるのは大変であり，プログラムを作成して対処する方がよいと判断し，そのプログラム作成のための情報をインターネットで検索した。幸い，ネット名「Nowral」氏（実際の氏名は現時点では分かっていない）の Web ページ

http : //homepage 3.nifty.com/Nowral/30_Mojikyo/30_Mojikyo.html

で，文字鏡の文字番号からフォント名と文字コードを求めるプログラムを Perl という言語で記述したものを掲載していたので，それを，以下のような C 言語のプログラムに書き直して用いた。

□リスト：文字鏡の文字番号からフォント名と文字コードを求めるプログラム

```
void MJ 2 FSJ(UINT mj, char *fcls, UINT *sjcodep)
{

  UINT blockno, charno, stradr, v, fno ;

  mj －－ ;

  fno = (mj / 5640) + 101 ;
  if (fno > 141) fno += 59 ;
  sprintf (fcls, "%3 d", fno) ;

  blockno = (mj / 94) % 60 ;    / / 区単位
  charno = (mj % 94) ;          / / 点番号
  if (blockno < 30) {  / / 第一水準
    stradr = 0x889F ;
  } else if (blockno < 45) {  / / 第二水準
    stradr = 0x989F ; blockno －= 30 ;
  } else {
    stradr = 0xE040 ; blockno －= 45 ;
  }

  blockno += (stradr >> 7 ) ;
  if ((blockno % 2 ) == 0) {
    charno += 0x40 ;
    if (charno >= 0x7F) charno++ ;
  }else {
    charno += 0x1F ;
  }

  v = (blockno << 7) + charno ;

  *sjcodep =  v ;
}
```

Ⅳ．Unicode 文字の扱い

4.1. Unicode 文字の取り扱い

Ⅲで述べたように追加漢字には，文字鏡のフォントを用いること，つまり，文字については，「シフト JIS 文字＋文字鏡文字」ということにし，システム開発およびシステムを利用しての辞書データの入力を進めてきた。もちろん，Windows 2000 からは，Unicode 文字のフォントなどが，デフォルトでインストールされるようになったことは，分かってはいたが，我々としては次の 2 つの理由により追加漢字には，文字鏡文字を用いることを継続することとしていた。

(1) Windows 2000 に備わっていた Unicode 文字は，あまり綺麗ではない。

なお，現在の Windows XP においても，あらかじめ備わっている Unicode 文字は，あまり綺麗ではない。

(2) Unicode での漢字の数は約 6 万文字程度，一方，文字鏡での漢字数は 8 万文字程度（現在は 12 万文字程度）であり，Unicode 文字を利用するとしても文字鏡の文字を利用せざるを得ないことがある。実際，既に追加漢字としている文字で，Unicode には含まれていないものもある程度の数存在した。逆に，文字鏡文字は Unicode 文字を含んでいるとのことである。

しかし，次のような状況から，Unicode にも対応した方がよいと判断することになり，現在では Unicode 文字を扱えるようにしている。

(a) 辞書データの中には，中国語の辞書を参考にしているところがあるが，そのような場合に，入力を中国の人に依頼していたが，送られてきたファイルでは Unicode の文字コードが用いられていた。

もちろん，その中で用いられている文字のうち，シフト JIS に存在するものは，シフト JIS に変換し，シフト JIS にないものを文字鏡文字に置き換えるという方法もある。しかし，シフト JIS にないものを洗い出すことも大変な作業であると思われ，データ入力の作業を長期間中断することなく進めるには，Unicode のまま扱うようにする方がよいと判断した。

(b) 開発したシステムは,満州文字を扱わないような,通常のテキストエディタとしても利用できるものとしても作成し，プログラム開発，HTMLやXMLなどのファイル作成のために用いていたが，XMLではUnicodeでのファイルとすることが標準で，シフトJISは特別な場合としての扱いとなっているようである。実際，インターネットで入手したサンプルのXMLファイルなどでは，Unicodeのものも多いし，Javaでの処理などを行う場合に，Unicodeのファイルとしていた方がいいと思われることも多々あった。

4.2. Unicode文字を扱うシステムの開発

4.2.1. 扱う文字コード体系

シフトJISの漢字コードとUnicodeの漢字コードは，重複している部分があるので，フォント切り替えなどの情報を含まない，単なるテキストファイルとして扱う場合は，同じファイルの中で用いることはできない。上に述べた理由で,満州語辞書のためにはUnicode文字を扱えることにする一方,通常のエディタとして利用する場合は,シフトJIS文字を扱えることも必要である。したがって，開発したシステムは,「シフトJIS文字＋文字鏡文字」，あるいは「Unicode文字＋文字鏡文字」のファイルを扱えるものとした。そして，読み込み時と，保存時は同じ文字コード体系でなくてもよいこととした。たとえば,「シフトJIS文字＋文字鏡文字」のファイルを読み込み，編集し,「Unicode文字＋文字鏡文字」のファイルとして保存することも可能なものとした。ただし，逆に「Unicode文字＋文字鏡文字」のファイルを読み込み，編集したのち,「シフトJIS文字＋文字鏡文字」のファイルで保存する場合は，Unicodeにはあるが，シフトJISに無い文字については，チェックし,「*??*」で置き換えることを警告し，保存するか保存を中止するか選択できるようにしている。

Unicodeの文字コード体系にはUTF-8，UTF-16 LE，UTF-16 BE，UTF-7，UTF-32など複数のものがあるが，多くの場合に用いられているのはUTF-8，UTF-16 LE，UTF-16 BEであるので，これらについてのみ扱うようにした。

Unicodeの場合，ファイルの先頭部に，次のようなBOM（Byte Order Mark）と呼ばれるファイル内の文字コード情報を記したものを用いることがある。(次のWebページ参照：

http : //homepage1.nifty.com/nomenclator/unicode/ucs_utf.htm,

http : //www.birdport.jp/Others/unicode/)

なお，ファイルが小さくUTF-8にしかない文字コードが用いられていないときは，BOMがないとシフトJISとの区別がつかないことがある。

○UTF-8

・BOMは付けても付けなくてもよい　　BOMはEF BB BF

・英数字（ASCII文字）は1バイト，漢字など英数字以外は2バイト

○UTF-16 BE

・BOMは付けても付けなくてもよい　　BOMはFE FF

・文字は2バイト（文字列ABCは16進数で00 41 00 42 00 43の順）

○UTF 16-LE

・BOMは必ず付ける　　BOMはFF FE

・文字は2バイト（文字列ABCは16進数で41 00 42 00 43 00の順）

UTF-16 LEとUTF-16 BEは，文字のコード長は2バイトで同じであるが，その上位バイトと下位バイトの順序が逆になっている（UTF-16 LEは下位バイトが先，UTF-16 BEは上位バイトが先)。なお，UTF-8の漢字のコードとUTF-16 LE，UTF-16 BEの漢字コードは異なるものである。

そこで，本編集サブシステム（エディタ）では，次の理由により，プログラム実行中のメモリ上でのテキスト管理にはUTF-16 BEを用いることとした。シフトJIS，あるいはUTF-8などの文字コードのファイルの場合でも，読み込み時にUTF-16 BEに変換し，プログラムのメモリ上ではUTF-16 BEのコードで扱うということである。

○　そもそもシフトJISの文字にはなく，Unicodeに含まれる文字を扱う必要のために作成しているので，Unicodeを用いる。

○　パソコンは，128 MB以上（現在では512 MB～1 GBである）の充分大

きな実メモリを備えているので，多少メモリを多くとる場合があるとしても，１文字２バイトと固定している方がプログラムを作成しやすい。実際，シフトJISの２バイト文字と英数字の１バイト文字が混在していたこれまでよりも，プログラムは簡単になったところも多い。

○　UTF-16 LEとUTF-16 BEについては，どちらにしてもよかったが，上位バイトが先にある方が自然であり，プログラム作成時に間違いを犯しにくいということが期待できる。

なお，既に述べたように，ファイルに書き出し・保存するときは，どの文字コード体系とするかを指定できるようにしている。

4.2.2. Unicode用のフォントファイル

Unicode　Consortiumの Webページ（http : //www.unicode.org/）で示されている漢字－いわゆるCJK（China, Japan, Korea）エンコーディングー－に関するものは「East Asia Scripts」の分類のところで，次のように示されている（下記はコード順ではなく，Webページ掲載順）。

・4E00～9FBF　：CJK Unified Ideographs
・3400～4DB5　：CJK Unified Ideographs Extension A
・20000～2A6DF　：CJK Unified Ideographs Extension B
・F900～FAFF　：CJK Compatibility Ideographs
・2F800～2FA1F　：CJK Compatibility Ideographs Supplement
・3190～319F　：Kanbun（漢文用）
・2E80～2EFE　：CJK Radicals Supplement（偏と旁）
・2F00～2FDF　：Kangxi Radicals
・31C0～31EF　：CJK Strokes
・2FF0～2FFF　：Ideographic Description Characters

さらに，「Japanese-specific」として，次のものも示されている。

・3040～309F　：Hiragana（ひらがな）
・30A0～30FF　：Katakana（カタカナ）
・31F0～31FF　：Katakana Phonetic Extensions
・FF00～FFEF　：Halfwidth and Fullwidth Forms（半角文字）

上記の文字コード体系のうち，20000～2A6FDの文字は現在画面表示の方法が不明であるので，編集サブシステムでは扱っていないが，実際この部分の文

字はまだ出現していないし，出現しても文字鏡文字で対応する予定であり，特に問題とはならないと思っている。また，F900～FAFF の文字については，既に満州文字と文字鏡文字に F000～FAFF までを割り当てていることもあり，用いないこととした。この部分についても，現在のところこの部分の文字は出現していないし，文字鏡文字で対応することとすれば，これも問題ないと思っている。

Unicode Consortium は，文字とコードの対応を規定しているが，具体的なフォントファイルを提供はしていないようである。したがって，実際に文字を表示・印刷するためには適切なフォントファイルが必要であり，MS-明朝フォントファイルを用いた。

日本語・満州語辞書では中国語での表記も入れているが，その場合の漢字は，繁体字と呼ばれているものである。繁体字フォントは台湾で利用されている漢字であるが，日本で用いられている漢字と同じものが多い。幸い MS-明朝フォント，JS-平成明朝体フォントは繁体字フォントである。一方，現在，中国大陸で用いられている漢字は簡体字フォントと呼ばれているものである。繁体字と簡体字の中国文字のコードは，繁体字：A140～F9F0，簡体字：A1A0～F7F0 であり，重複している部分があること，また，フォントファイルは，繁体字には MingLiU，PMingLiU が，簡体字には MS Hei，MS Song，Sim Sun があることが分かったので，現在進めている日本語・満州語の辞書作成以外では簡体字を用いることがあるかもしれないことを考慮し，編集サブシステムでは，繁体字と簡体字を切り替えて使用できるようにしている。

先に述べたように Unicode のフォントファイルは Windows に付属している MS-明朝フォントファイルを用いていたが，その後，日本語ワープロソフトの一太郎に付属している JS-平成明朝体 W３フォントの方が綺麗であるので，現在ではそちらを用いている。ただし，最終的な辞書の印刷で，JS-平成明朝体 W３フォントを用いるか，あるいは文字鏡文字に置き換えるのかは未定である。文字の綺麗さと Tex で文字鏡文字と Unicode 文字のどちらが扱いやすいかを判断して決めることにしている。

4.2.3. Unicode 文字，満州文字，文字鏡文字の表示

Visual Studio 6 での C 言語のプログラムで，Unicode の文字を画面に表示する方法は，参考資料(3)の Web ページなどを参考にし，大まかには，次のようにすればよいことが分った。

○ファイルの最初の方に以下の２行を入れる。

```
#defint UNICODE
#define _UNICODE
```

○ヘッダファイルをインクルードするところに，次の行を入れる。

```
#include <TCHAR.h>
```

○文字列部分の変更

シフト JIS のときは，"ABC" のように"（ダブルクオート）で囲んでいた文字列を，TEXT（"ABC"）あるいは_T（"ABC"）とする。

○Unicode フォント（JS 平成明朝体 W 3）の設定

```
hFJSMin = CreateFont (0, 0, 0, 0, FW_NORMAL, 0, 0, 0,
                DEFAULT_CHARSET, OUT_DEFAULT_PRECIS, CLIP_
                DEFAULT_PRECIS,
                DEFAULT_QUALITY, FIXED_PITCH, TEXT ("JS 平成
                明朝体 W 3"));
SelectObject (hdc, hFJSMin);
```

○文字列を表示する関数 TextOut（　）の記述

```
TextOut (hdc, X, Y, unistr, wcslen (unistr));
```

ただし，編集サブシステムでは，Unicode 文字だけでなく，フォントを作成した満州文字，および文字鏡文字を表示する必要もあるが，上記のようにするとこれまで表示できていた満州文字フォント，文字鏡文字フォントの文字が表示できなくなった。

インターネットで Visual Studio を用いての WIN 32 API プログラム開発において Unicode を表示する方法を調べたところ，[Unidoe 関連 Web ページ]の(2)などでは，次のようなことが記されていた。

Visual Studio では，プリプロセッサの処理として，「#define UNICODE」

などでの指定があれば，プログラム中のTextOutの呼び出しは関数TextOutWの呼び出しに，指定がなければ関数TextOutAの呼び出しに置き換えられる。また，同じプログラムのあるところでは，従来の文字体系のフォントで表示し，別のところではUnicode文字体系のフォントで表示するには，それぞれ，プログラム中で，関数TextOutAあるいはTextOutWを用いればよい。

そこで本編集サブシステムでも，満州文字や文字鏡文字は関数TextOutAを呼び出し，Unicode文字は関数TextOutWを呼び出すように変更したが，現在まで特に問題は生じていない。

V．さいごに

本システムは，最初に相談を受けてから，長期間のプロジェクトとなったために，用いるパソコン機種も日本電気のPC-9801シリーズからDOS/Vパソコンへ変更せざるを得なくなったことに対応した。またOSもMS-DOSからWindows 95，Windows 98，Widnows Meの16ビットWindowsから，Windows 2000，Windows XPの32ビットWindowsなどに変遷し，それに応じた対応をしてきた。さらに，Windows 2000，Windows XPではUnicodeも用いられるようになった。しかし，OSをWindows 2000にした時点ではUnicodeに対応する必要がなかったが，現在では対応の必要が生じてきた。本システムを用いての辞書の編纂が終了するまで，今後もこれまでと同様に，システム側の変化の対応，および本システムを利用する側のための必要性，あるいは便宜のために，今後とも対応が続くものと考えている。

なお，Unicode文字体系の領域のうちF900～FAFFを用いているものが出現した場合については次のように考えている。現在本システムで用いている満州文字と文字鏡の文字数であれば，満州文字と文字鏡文字のコードをF900～FAFFを用いないように移すことによりUnicodeの文字コードをそのまま用いるようにすることも可能である。つまり，満州文字と文字鏡文字のコード割り当てを変更すれば，Unicode文字と，満州文字，文字鏡文字を共存させる文字

コード体系とすることは可能である。しかし，辞書作成の進行とともに，利用する文字鏡文字が徐々に増加してきたこともあり，最終的に必要な文字鏡文字を入れてもF900～FAFFを用いなくて済むかどうか不明であること，最終的な印刷をUnicodeのフォントで行うか，それとも文字鏡文字フォントで行うかが未定であることなどから，現時点で文字コード領域を移動することは考えていない。

参　考　資　料

本田［2003］「日本語・満州語の辞書作成のための補助システム（Ⅲ）」

Unicodeなどの文字コードに関連したwebページ

(1) 一般的な文字コードについて
- フォント・多国語処理：http : //makotowatana.ld.infoseek.co.jp/font.html

(2) Unicode文字体系についての説明
- UCSとUTF：http : //homepage1.nifty.com/nomenclator/unicode/ucs_utf.htm
- Unicode：http : //homepage3.nifty.com/Nowral/31_Unicode/31_Unicode.html
- 文字コード，標準化について：http : //ash.jp/code/

(3) UnicodeとシフトJISの文字の表示
- UNICODEプログラムの作り方（入力＆表示編）：
  http : //www.honet.ne.jp/~tri/program/unicows02.html
- UNICODEプログラムの書き方：
  http : //www.246.ne.jp/~y-ookubo/program/tips/unicode.html
- Unicode関数：http : //wisdom.sakura.ne.jp/system/winapi/win32/win151.html

(4) 中国語の繁体字と簡体字
- 中国語の文字コード：http : //ash.jp/code/code_zh.htm