Supporting the Character Sets of Japanese Kanji and Korean Hangul in the ADABAS/NATURAL System

Masahiro Shimada¹, Hideki Nishimoto², Takashi Ishizaka¹, Andreas Schütz³, Yong-Soo Kim⁴ and Yoshioki Ishii⁴

¹) Software AG of Far East Inc., JAPAN
²) University of Library and Information Science, JAPAN
³) Software AG, WEST GERMANY
⁴) Penta Computer Korea Co., Ltd., KOR

ABSTRACT

Software AG of Far East (SAGFE) has established various system environments peculiar to Japanese use since Japanese Kanji was supported on ADABAS for the first time in 1978.

The supporting of Japanese language, which started by putting the character strings on a special Kanji printer, has recently been improving with the development of terminal equipment and controllers.

This paper describes the progress of our supports for Japanese language at SAGFE and Software AG (SAG), West Germany and a study on Korean Hangul done by SAG and Penta Computer Korea as well as Japanese Kanji. Additionally, the method called DBCS (double-byte character set) support is proposed. We discuss the problems and solve those problems by adapting the fourth generation language, NATURAL, as a SAG product.

1. Supporting Japanese Kanji on ADABAS

The Japanese language borrowed extensively from the Chinese Hanzi by way of Korean to write their own language around 1,500 years ago. Later, the characters came to symbolize native Japanese words similar in meaning to that of the Chinese.

Modern Japanese is written as a mixture of ideograms, Kanji, and native phonetic letters, Kana. The Kana phonetic alphabet exists as Hiragana and Katakana, which serve different purposes and differ stylistically. A typical passage of Japanese writing contains Kanji, Hiragana, and perhaps also Katakana. Both Kana syllabaries consist of 46 basic symbols each, and Kanji characters are limited to about 2,000 symbols for official and daily use.

Fig. 1-1 Japanese character and code representation

In the early Japanese computer market, there was intense interest in the establishment of a system to handle Kanji. However, it took a long time and a vast sum of money to create compatibility with existing systems and to develop devices peculiar to Japanese use. Generally one byte is used to denote a character, but the eight-byte byte provides for as many as 256 characters. Japanese Kanji must be represented by codes that are usually two bytes long because there are many different kinds of characters (Fig. 1-1). This point is pressing computer manufacturers and vendors in Japan to greatly revise standard systems.

A great number of Kanji typing methods have been developed to date, all of which fall into three main...
categories: two-dimensional selection array, coding scheme, and phonetic conversion [1]. Currently phonetic conversion is the most popular method for general users in Japan by reason of its having so few rules to learn, its use of English-typing skills, and others.

The history of the supporting Kanji on the ADABAS system in Japan can be divided into four stages as follows: (Fig.1-2)

---

**Stage I) Putting Kanji characters on a special printer by batch processing**

In 1978, we supported the Japanese Kanji first on the law retrieval system developed with an ADABAS system at the Administrative Management Agency (the Management and Coordination Agency at present). The system is for retrieving the texts of laws and regulations, which the user does by specifying key words. The texts are then printed out on a printer. There was not any terminal equipment displaying the Kanji characters on the market; therefore, putting on a special Kanji printer after the retrieval by batch processing mode. To make the system more flexible, an option was built in allowing for the form of the Kanji text to be freely adjusted.

**Stage II) Supporting early Kanji terminals**

As soon as the early Kanji-supported terminal came onto the market in 1980, we tried using the user-friendly query system ADASCRIPT on ADABAS with the terminal. The fourth generation language NATURAL (Version1.1), to develop application programs on ADABAS, was released at the same time. Their terminal drivers were ought to be upgraded to process Kanji data on the screen.

Kanji terminals with different specifications were made by different manufacturers, so we had to cope with each model separately.

In 1981, we succeeded at Kyodo Oil Company in adopting a method in which a map with Kanji characters and ruled lines are created in advance. The map is synthesized with the NATURAL screen at run time in order to completely support "the Kanji ruled-line map facility" for the Fujitsu terminal. In the ensuing year the Hitachi terminal also was supported by this method.

**Stage III) Providing the phonetic conversion typing method**

Typing Kanji characters into the system is the job of the technical operators, done mainly by typing the Kanji codes or with a Kanji tablet.

In 1982, SAGFE succeeded in equipping the NATURAL system with a phonetic conversion method. The typist first enters its Hiragana spelling of the text directly enclosed with apostrophes, then the word-unit homophones are searched with the dictionary. After they appear on the screen, the operator can choose the number of the appropriate word. This is the same popular method used for personal computers in Japan. Additionally, our method can be implemented on all non-intelligent terminals because it does not require a special dictionary or an additional processor on the terminal.

---
Meanwhile, IBM started to support Japanese Kanji. Their system handled complex screen images with Kanji characters with difficulty, since it supported only the method identifying the Kanji mode as a field attribute. Therefore, it was not broadly accepted in the market place.

Stage IV: Developing the environment of the DBCS support

In 1982, an important problem arose when the new version NATURAL (V1.2) was released because it was equipped with the Map Editor facility for designing the direct output image on the screen.

The processing of the screen I/O is very difficult using our method in this case because the length of physical data going into the terminal is greater by some bytes of the control-codes than the length of characters displayed on the screen.

A new innovation of SAGFE was the development of the Kanji Proto-typing Interface (KAPRI) product with a KMAP facility which provides for adjusting the length of the data stream. Additionally, we succeeded in implementing KAPRI for the IBM 3278 terminal and it contributed to the NATURAL system's hot sales.

In 1984, IBM developed the virtual terminal emulator for the personal computer. This facility adapts the same Kanji shift code as Japanese products use. Since then, Kanji character data has been mostly put into the computer through the phonetic conversion processor on an intelligent terminal or on a workstation. Consequently, mainframes mainly undertake data manipulations after the Kanji has been put in.

The first version of NATURAL2 was formally released in 1987. Some of its screen operations were strongly reinforced. In the same year, SAGFE developed and released the KAPRI2 [3] product corresponding to the NATURAL2 system. Recently, SAG has decided to design and develop the double-byte character set (DBCS) products in order to completely support all the multi-character sets in the world. The specifications are discussed in Chapter 4.

There are, besides IBM, several manufacturers in Japan developing products compatible with IBM software. The products of the operating system, the TP monitor, and the terminal supported by the ADABAS/NATURAL system are shown in Fig.1-4. This demonstrates the high adaptability of ADABAS/NATURAL and the strength of SAGFE's support.

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>OS</th>
<th>TP monitor</th>
<th>Terminal (Vosa)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM</td>
<td>MVS, MVS/XA</td>
<td>VM/CMS</td>
<td>3270, 3620</td>
</tr>
<tr>
<td></td>
<td>VSE</td>
<td>COM-PLETE</td>
<td>3279</td>
</tr>
<tr>
<td></td>
<td>DOS, VSE</td>
<td>TSS, CICS</td>
<td>3279</td>
</tr>
<tr>
<td></td>
<td>VM/CMS</td>
<td>IMS</td>
<td>3279</td>
</tr>
<tr>
<td>FUJITSU</td>
<td>OS/W/4, MSF</td>
<td>OS/W/4, FSP</td>
<td>5051, 5061</td>
</tr>
<tr>
<td></td>
<td>OS/W/4, MSF</td>
<td>OS/W/4, FSP</td>
<td>5051, 5061</td>
</tr>
<tr>
<td>HITACHI</td>
<td>VOS 3</td>
<td>TESS, ADM, DOCCII, DOCCIII, TIMES V</td>
<td>5051, 5061</td>
</tr>
<tr>
<td></td>
<td>VOS 2</td>
<td>DOCCII, DOCCIII, TIMES V</td>
<td>5051, 5061</td>
</tr>
<tr>
<td></td>
<td>VOS 1</td>
<td>DOCCII, DOCCIII, TIMES V</td>
<td>5051, 5061</td>
</tr>
<tr>
<td>MITSUBISHI</td>
<td>GOS/VS</td>
<td>TESS, LIXES</td>
<td>M4375</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F4 with emulators</td>
<td></td>
</tr>
</tbody>
</table>

Note 1: If a terminal can be connected to other manufacturers' OSs and TP monitors, then the terminal is able to work under those OSs and TP monitors, i.e. terminals can be supported independent of OSs and TP monitors.

Fig.1-4 ADABAS/NATURAL environments supported in Japan
2. An outline of Korean Hangul

In Korean, the use of Hanzi is gradually being phased out, although the Korean language was also deeply affected by Chinese Hanzi characters. Now Korean is commonly written in a pure phonetic alphabet, called Hangul, with no Hanzi at all. Hangul was invented in 1446 during the reign of Sejong, the fourth King of the Yi dynasty. It is the only script known to have been designed by a committee.

This phonetic alphabet has 24 letters, consisting of 10 vowels and 14 consonants. The letters are designed to be grouped into square clusters of two, three, or occasionally four letters. Each cluster represents one spoken syllable of the language. The number of characters required for general use come to about 2,000 clusters (Fig.2-1).

A Hangul cluster may come in different shapes and positions, depending on its surrounding context. A computerized Hangul typing system can have a single generic key or a Romanized phonetic symbol for each Hangul letter. The computer is relied on to decide the proper graphical form and position for each letter in relation to the letter or letters surrounding it. Computer display is very good for "movable" typing because each new letter usually requires revising the form and placement of letters that have previously been typed [1] (Fig.2-2).

3. Problems of double-byte character set support

The common problems on the support of handling multi-characters are discussed in this chapter. The first problem is to create several code notations representing the same character set. Manufacturers naturally establish the original code specification before standards are completely set in the country for the computerized code notation of each character. The two types of code tables of Japanese Kanji are shown in Fig.3-1.
Although the IBM Kanji code is organized as an expansion of EBCDIC, the JEF code by Fujitsu and the KEIS code by Hitachi adopt the shifted JIS code. In an environment of distributed processing, the character strings belonging to the same character sets, even if each is based on different code notations, must be executed correctly in comparison or move operations. Consequently, a data transformation function may be additionally required, and this is seen to be a very important role of the database management system.

Second, there may be many difficulties in handling shift control codes. In the terminal data stream, double-byte characters are generally distinguished from single-byte characters by checking the beginning and end of strings with the Shift Out/Shift In (SO/SI) control code.

Each manufacturer has its own specifications of the SO/SI codes, and this is an important problem in the distributed processing environment.

The database management system should also be able to completely transform the control codes used by every site (Fig.3-2). Additionally, the internal procedures of the data manipulations, Search, Move, Edit and others should be revised with various options to keep the results consistent (Fig.3-3).

![Fig.3-2 ADABAS/NATURAL distributed environment](image)

4. The DBCS support on the ADABAS/NATURAL system

The double byte character set (DBCS) support of ADABAS/NATURAL consists of two products, the ADABAS/NATURAL code standardization and the DBCS facility.

The former product is designed for correctly processing data manipulations, even among different code notations belonging to the same character set.

In the ADABAS/NATURAL world, all representation of character strings would usually conform to the standard code notation established for each character set in advance (Fig.4-1). DBMS should keep environments of every site and give proper code transformation during only the I/O communication. Consequently, the internal procedures of the data manipulation can keep the results consistent; additionally each site does not need any additional transformation program or huge table.
The latter product DBCS facility has been developed so that all NATURAL operations can completely accept every variable, name, and data value described in any double-byte character as well as single-byte character defined as EBCDIC or ASCII. The DBCS facility includes the following strong points:

a. Reinforced Editors

The NATURAL offers four types of editors [2]. The Edit commands affected by supporting DBCS completely are the Split, the Join, the Scan and the Change of any NATURAL editor mode. All cursor-oriented commands (Join, Split) should be checked to see if the cursor is in the middle of a DBCS field. If so, the cursor has to be on an even byte boundary controlled by the terminal. Additionally, in the split function, the split fields have to be enclosed by SO/SI codes at the end of the split operation.

The Join command checks to see if the two new adjacent fields are both enclosed in SO/SI codes. If so, the terminating SI of the leading field and the starting SO of the following field are eliminated (see Fig.3-3).

The Scan and Change commands check to see if the argument strings are enclosed in SO/SI codes and will then locate targets only if the analysed strings are enclosed in SO/SI codes as well.

b. Names written in DBCS

The names of programs, data areas, database areas and others used in the NATURAL environment can be written in DBCS. It is important for the synthesized data management, documentation, and program maintenance. If each name cannot exceed n bytes, this means that names coded in DBCS can have up to n/2 characters, including SO/SI codes. These names always have to be enclosed in SO/SI codes to allow NATURAL editors, the NATURAL List functions and the Scan functions to work correctly.
c. DBCS field type

There are two ways for NATURAL to recognize the DBCS field. If a field value is represented in DBCS mode, it starts with the SO code and is terminated with a SI code. These SO/SI codes are part of the field value. Field values passed to or from NATURAL do not need special considerations, with the exception of the NATURAL editors and all possible value splits on an odd-byte boundary.

The other possibility to identify field values in DBCS representation is the DBCS field type by the new print mode format 'D'. This format can be specified as a new value to the session parameter, PM (Print mode) = D, which indicates for the given field that the data of that field is completely in DBCS mode. At terminal I/O time NATURAL automatically will add the attribute for these fields to switch the terminal to the correct set. This technique will save two bytes for each field in the database but requires the knowledge of which data is in DBCS mode to process them correctly. The NATURAL data description module (DDM) will accept the specification of 'PM = D' as part of the ADABAS field description.

d. SO/SI preservation during data manipulations

The following main NATURAL statements for data manipulation can also deal with DBCS characters completely:

1. Separate statement
   The Separate function is used for dividing one character string into several other character strings. This statement checks the field contents for a starting SO. If found, all scanning will be done in two-byte steps; the resulting strings will all be enclosed in SO/SI codes.

2. Compress statement
   Compress is the function used for joining two character strings. This statement checks if two joining strings end and start with SO/SI codes and will eliminate them to form one complete DBCS character string. If the delimiter character used is space and the two joining strings are in DBCS mode, the SO/SI codes are removed from the end of the first field as well as from the start of the second field.

3. Move statement
   A check has to be performed if the source field contains data starting with SO when data is moved from one field type alphanumeric into another alphanumeric type field. If so, the resulting field value has to be enclosed in SO/SI codes.

5. Conclusion

So far, the handling of double-byte characters in the ADABAS/NATURAL system environment has been supported with an additional product, "KAPRI"[3], by SAGFE. However, it is for just putting the characters on specially designed terminal displays or printers used in Japan.

Recently we have planned to design a complete DBCS support and build it into the nucleus of the ADABAS/NATURAL system in order to deal with double-byte characters correctly for various data manipulations.

It may become easier to build applications for handling other multi-characters sets, in addition to Kanji or Hangul, if the character data are managed on the ADABAS/NATURAL standard code system and operated through the DBCS facility we have presented.

References