SUPPORTING THE CHINESE, JAPANESE, AND KOREAN LANGUAGES IN THE OPENVMS OPERATING SYSTEM

By Michael M. T. Yau

ABSTRACT

The Asian language versions of the OpenVMS operating system allow Asian-speaking users to interact with the OpenVMS system in their native languages and provide a platform for developing Asian applications. Since the OpenVMS variants must be able to handle multibyte character sets, the requirements for the internal representation, input, and output differ considerably from those for the standard English version. A review of the Japanese, Chinese, and Korean writing systems and character set standards provides the context for a discussion of the features of the Asian OpenVMS variants. The localization approach adopted in developing these Asian variants was shaped by business and engineering constraints; issues related to this approach are presented.

INTRODUCTION

The OpenVMS operating system was designed in an era when English was the only language supported in computer systems. The Digital Command Language (DCL) commands and utilities, system help and message texts, run-time libraries and system services, and names of system objects such as file names and user names all assume English text encoded in the 7-bit American Standard Code for Information Interchange (ASCII) character set.

As Digital's business began to expand into markets where common end users are non-English speaking, the requirement for the OpenVMS system to support languages other than English became inevitable. In contrast to the migration to support single-byte, 8-bit European characters, OpenVMS localization efforts to support the Asian languages, namely Japanese, Chinese, and Korean, must deal with a more complex issue, i.e., the handling of multibyte character sets. Requirements for the internal representation, input, and output of Asian text are radically different from those for English text. As a result, many traditional ASCII programming assumptions embedded in the OpenVMS system are not valid for handling Asian multibyte characters.

Since the early 1980s, Digital's engineering groups in Asia have been localizing the OpenVMS system to support Asian languages. The resultant Asian language extensions allow Asian-speaking users to interact with the OpenVMS system in their native languages. These extensions also provide a platform for developing Asian applications. This paper presents a high-level overview of the major features of Chinese, Japanese, and Korean support in the OpenVMS operating system and discusses the localization approach and techniques adopted.

ASIAN LANGUAGE VARIANTS OF THE OpenVMS OPERATING SYSTEM

The following five separate Asian language variants of the OpenVMS operating system are available in the Pacific Rim geographical area:

LANGUAGE	COUNTRY	OpenVMS VARIANT
Japanese	Japan	OpenVMS/Japanese
Chinese	People's Republic of China	OpenVMS/Hanzi
Chinese	Taiwan, Republic of China	OpenVMS/Hanyu
Korean	Republic of Korea (South Korea)	OpenVMS/Hangul
Thai	Thailand	OpenVMS/Thai

This paper covers the first four variants, omitting the Thai variant because of space limitations. Each Asian language variant of the OpenVMS system is designed to be installed and to run as a separate system. Currently, no provision exists to formally support multiple Asian languages simultaneously on a single OpenVMS system. Each variant provides a bilingual system environment of English and one Asian language. Such an environment, called Asian OpenVMS mode in this paper, supports ASCII and one multibyte Asian character set. The variants are available on the VAX and the Alpha AXP platforms with identical features. Throughout the paper, the generic name Asian OpenVMS variant denotes any of the Asian language variants of the OpenVMS operating system, regardless of the hardware platform.

To achieve full downward compatibility for existing users, applications, and data from the standard OpenVMS system, each Asian OpenVMS variant is a superset of the standard OpenVMS system. In fact, a user can operate in the standard OpenVMS mode, i.e., the 1-byte DEC Multinational Character Set (DEC MCS), on an Asian OpenVMS variant without noticing any difference in the functional behavior compared to a standard OpenVMS system. The components of an Asian OpenVMS variant are installed on a standard OpenVMS system in a manner similar to that of a layered product; files (executable images and other data files) are added and replaced on the standard OpenVMS system. In general, three types of components are supplied in an installation:

1. A standard OpenVMS component supplanted by an Asian localized version that includes the standard OpenVMS mode as a subset. At the process level, the user can set the component to run in either standard OpenVMS mode or Asian OpenVMS mode. The DCL and the terminal driver are examples of this type of component.

2. A standard OpenVMS component supplemented by an Asian localized version that runs only in Asian OpenVMS mode. Both versions of the component run simultaneously on the system. Examples are the TPU/EVE editor and the MAIL utility.

3. A new Asian-specific component created to provide functionality for Asian processing that does not exist in the standard OpenVMS system. An example of this type of component is the character manager (CMGR), which is discussed later in this paper.

OVERVIEW OF ASIAN WRITING SYSTEMS

Before looking at specific features of the Asian OpenVMS variants, this paper briefly reviews the Chinese, Japanese, and Korean writing systems. For a more detailed discussion of the differences among these writing systems, refer to Tim Greenwood's paper in this issue of the Journal.[1]

The Chinese Writing System

The Chinese writing system uses ideographic characters called Hanzi, which originated in ancient China more than 3,000 years ago. Each ideographic character (or ideogram) is a symbol made up of elementary root radicals that represent ideas and things. Some ideograms have very complex glyphs that consist of up to 30 brush strokes. Over 50,000 Chinese ideograms are known to exist today; however, a subset of 20,000 or less is typically sufficient for general use. Two or more ideograms are often strung together to represent more complex thoughts.

Ideographic writing systems have characteristics that are quite different from those of alphabetical writing systems, such as the Latin languages. For instance, the concept of uppercase and lowercase does not apply to ideographic characters, and collation rules are built on different attributes. The input of ideographic characters on a standard keyboard requires additional processing.

Two forms of Chinese characters are in use today: Traditional

Chinese and Simplified Chinese. Traditional Chinese is the original written form and is still used in Taiwan and Hong Kong. In the 1940s, the government of the People's Republic of China (PRC) launched a campaign to simplify the writing of some traditional Chinese characters in an effort to speed up the learning process. The resulting simpler set of Chinese characters is known as Simplified Chinese and is used in the PRC, Singapore, and Hong Kong.

The Japanese Writing System

The Japanese writing system uses three scripts: Chinese ideographic characters (called kanji in Japan), kana (the native phonetic alphabet), and romaji (the English alphabet used for foreign words). The kanji script commonly used in Japanese includes about 7,000 characters. There are two sets of kana scripts, namely, hiragana and katakana; each comprises 52 characters that represent syllables in the Japanese language. Hiragana is used extensively intermixed with kanji.
Katakana is used to represent words borrowed from other languages.

The Korean Writing System

The Korean writing system uses two scripts: Hangul (the native phonetic characters) and Hanja (Chinese ideographic characters). The Hangul script was invented in 1443 by a group of scholars in response to a royal directive. Each Hangul character is a grouping of two to five Hangul letters (phonemes) that forms a square cluster and represents a syllable in the Korean language. The modern Hangul alphabet contains 24 basic letters -- 14 consonants and 10 vowels. Extended letters are derived by doubling or combining the basic letters.

ASIAN CHARACTER SETS

During the early days of Asian language computerization when de jure standards did not exist for Asian character sets, individual vendors in the local countries invented their own local character sets for use in their Asian language products. Although most vendors have migrated to conform with the national standards, a variety of local character sets still exists today in legacy systems, thus creating interoperability issues. This paper reviews only the national standard character sets that are supported by the Asian OpenVMS variants.

National Standards

National standards bodies in each of the Asian Pacific geographies have established character set standards to facilitate information interchange for their local characters. For languages that use Han characters (which are large in number) in their writing scripts, the character set standards all share a similar structure, which is illustrated in Figure 1. Characters are assigned to a 94-row by 94-column structure called a plane. Each character is represented by a 2-byte (7-bit) value in the range of 0x21 to 0x7E. A plane, therefore, has a total of 8,836 code points available. Such a structure avoids the ASCII control code values, thus preventing conflicts with existing operating systems and communication hardware and software.

Figure 1 Code Structure in Asian Character Set Standards

```
       COLUMN
     1        94            ROW               COLUMN
    1+---------+    +-+--------------+  +-+--------------+
     |         |    |*|              |  |*|              |
ROW  |         |    +-+--------------+  +-+--------------+
     |         |         0x21-0x7E           0x21-0x7E
     |         |
   94+---------+             TWO-BYTE CODE STRUCTURE
       A PLANE
```

*Note that the first bit of each row and column can be either 0 or 1.

Japan. Japan was the first country to announce a 2-byte character set standard, the Code of the Japanese Graphic Character Set for Information Interchange (JIS C 6226-1978).[2] This standard has since been revised twice, in 1983 and 1990, and renamed JIS X 0208. The JIS X 0208-1983 standard includes 6,353 kanji characters divided into two levels, according to frequency of usage.[3] Level 1 has 2,965 characters, and level 2 has an additional 3,388 characters. This standard also includes complete sets of characters for hiragana and katakana, ASCII, and the Greek and Russian scripts -- a total of 453 characters. The 1990 revision, JIS X 0208-1990, added two characters to the standard.[4] An additional plane of kanji characters became standard in 1990 with the announcement of JIS X 0212-1990.[5]

Prior to the introduction of the 2-byte standards, Japanese systems that support katakana used the JIS X 0201-1976 standard for a 1-byte, 8-bit character set.[6] Today, there is still a demand to support this standard, in addition to the 2-byte standards, due to its pervasive use primarily in legacy mainframe systems.

People's Republic of China. In 1980, China announced a 2-byte standard, Chinese Character Coded Character Set for Information

Interchange -- Basic Set (GB 2312-1980).[7] Its structure, which follows that of the Japanese standard, includes two levels of Hanzi. Level 1 has 3,755 characters, and level 2 has an additional 3,008 characters. The standard also has 682 characters, including ASCII, Greek, Russian, and the Japanese kana characters. Subsequently, China has announced additional character set standards.

Taiwan, Republic of China. The Taiwanese national standard, Standard Interchange Code for Generally Used Chinese Characters (CNS 11643-1986) was first announced in 1986.[8] Again, the structure is similar to the Japanese and PRC standards. It defines two planes of characters with a total of 13,051 Hanzi, 651 symbols, and 33 control characters. The standard was revised in 1992 and renamed Chinese Standard Interchange Code (CNS 11643-1992).[9] An additional five planes were defined in this revision, adding 34,976 characters.

Republic of Korea (South Korea). The latest version of the Korean 2-byte character set standard is the Korean Industrial Standards Association Code for Information Interchange (KS C 5601-1987), announced in 1987.[10] This standard includes 2,350 precomposed Hangul characters, 4,888 Hanja (Chinese characters), and 352 other characters such as ASCII, the Hangul alphabets, Japanese kana, Greek, Russian, and special symbols.

User-defined Characters

Character set standards do not always encode all known characters of the writing scripts for which the standards are intended. For instance, when the total number of known characters exceeds the available code space, only subsets of the most common characters are included. In addition, new characters are invented over time to describe new ideas or objects, such as new chemical elements. The concept of user-defined characters (UDCs), sometimes known as gaiji in Japan, was introduced to address the user's need for characters that are not coded in a character set standard. Many computer vendors, including Digital, provide extended code areas for assigning UDCs and vendor-defined nonstandard characters. Attributes of these characters for various operations such as display fonts, collation weights, and input key sequence are often made available, e.g., by registering them in a system database. From an end-user and application perspective, UDCs should be processed transparently and in the same way as standard characters.

ASIAN OpenVMS SYSTEM OVERVIEW

From an operating system perspective, three basic issues need to be addressed to support Asian character processing, namely, internal representation, (i.e., how Asian characters are represented and stored inside the computer), basic text input, and output.

Internal Representation

Asian OpenVMS variants support the respective national standard character sets. To achieve full compatibility with existing ASCII data, each Asian OpenVMS variant simultaneously supports one multibyte Asian character set and ASCII. A variety of schemes can be used to combine multiple coded character sets. In general, the schemes fall into one of the following three types:

1. Shift code-based representation. In this scheme, the 1-byte code is combined with a 2-byte code by inserting shift control codes to switch between the two code sets. A 1-byte "shift out" control code changes the mode from 1- to 2-byte, while a 1-byte "shift in" control changes the mode back to 1-byte characters. This scheme is in common use in mainframes.

2. ISO 2022-based representation. The ISO 2022 Code Extension Techniques allow a designated character set to consist of two, three, or four 7-bit bytes in addition to the 7-bit sets.[11] The only requirement is that all bytes of a character have the same high-order bit setting (all 0 or all 1). A simple scheme of simultaneously supporting ASCII and one 2-byte character set can be achieved by statically designating ASCII to G0 and invoking it to graphics left (GL) and designating a local 2-byte set (e.g., one of the Chinese, Japanese, or Korean sets) to G1 and invoking it to graphics right (GR). The resulting mixed 1-byte/2-byte representation is shown in Figure 2.

   The high-order bit of each 8-bit byte provides self-identifying information for the local 2-byte set. This scheme can be further extended to include two additional character sets by statically designating them to G2 and G3 and invoking them by the single shift codes SS2 and SS3. The Extended UNIX Code (EUC) scheme employs this additional extension.

Figure 2 Example of an ISO 2022-based Representation That Combines Multiple Coded Character Sets

```
     FIRST BYTE         SECOND BYTE

  +-+-+-+-+-+-+-+-+
  |0|             |
  +-+-+-+-+-+-+-+-+
     7-BIT ASCII

  +-+-+-+-+-+-+-+-+  +-+-+-+-+-+-+-+-+
  |1|             |  |1|             |
  +-+-+-+-+-+-+-+-+  +-+-+-+-+-+-+-+-+
           14-BIT JIS/GB/KS
```

3. Shift range-based representation. This scheme, a hybrid of the previous two schemes, is used by the "Shift JIS Code" on PC-based systems in Japan. Bytes with codes 0 to 127 are interpreted as 1-byte ASCII, codes 160 to 191 and 192 to 223 are interpreted as 1-byte katakana (as specified by the JIS X 0201 standard), and codes 128 to 159 and 224 to 255 are combined with the byte that follows to form a 2-byte code that is interpreted as a kanji character (as specified by the JIS X 0208 standard). This scheme allows more single-byte characters to be represented at the expense of the number of 2-byte characters allowed.

Asian OpenVMS variants employ the ISO 2022-based representation for Digital's Asian code sets (the DEC Asian code sets) and are named respectively DEC Kanji, DEC Hanzi, DEC Hanyu, and DEC Hangul for the Japanese, Simplified Chinese, Traditional Chinese, and Korean character sets. This encoding scheme maintains full downward compatibility with all existing ASCII software and data. In particular, a string or record that consists of only ASCII characters has the form of simple ASCII. Because there is no need to keep state information about the data, this scheme simplifies processing, when compared to the shift code-based scheme. However, without explicit support for coded character set designation, simultaneous support for Chinese, Japanese, and Korean is not possible.

To support UDCs, each DEC Asian code set contains an extended code area for their assignment. The high-order bit of the second byte no longer has to be set; thus, an additional 94 by 94 plane of code positions is available. The disadvantages of this approach are that synchronizing a character boundary requires scanning forward from the beginning of the string and that the second byte can now conflict with the ASCII characters.

The DEC Asian code set internal representation corresponds to mapping a character plane (94 by 94) to one of the (1,1) and (0,1) quadrants of the 2-byte code space in Figure 3. The exact mappings of individual DEC Asian character sets supported by Asian OpenVMS vary. Table 1 provides a summary of the common code range assignments.

Figure 3 DEC Asian Code Set Internal Representation

```
                                SECOND BYTE
```

```
                00  21           80   A1           FF
                +---+------------+---+------------+
                |               CONTROLS          |
              21+   +------------+   +------------+
     CONTROLS   |   |            |   |            |
              \ |   |            |   |            |
               \|   |   (0,0)    |   |   (0,1)    |
                \   |            |   |            |
                |\  |            |   |            |
      FIRST   80+   +------------+   +------------+
      BYTE      |               CONTROLS          |
              A1+   +------------+   +------------+
                |   |            |   |            |
                |   |            |   |            |
                |   |   (1,0)    |   |   (1,1)    |
                |   |            |   |            |
                |   |            |   |            |
              FF+---+------------+---+------------+
```

Table 1 Summary of DEC Asian Code Range Assignments

CODE RANGE	DEC KANJI	DEC HANZI	DEC HANYU	DEC HANGUL
(0xxxxxxx)	ASCII	ASCII	ASCII	ASCII
(1xxxxxxx 1xxxxxxx)	JIS X 0208	GB 2312-1980	CNS 11643-1986(1)*	KS C 5601-1987
(1xxxxxxx 0xxxxxxx)	UDC	UDC	CNS 11643-1986(2)!	--
(000xxxx)	C0 Control	C0 Control	C0 Control	C0 Control
(100xxxx)	C1 Control	C1 Control	C1 Control	C1 Control

Notes:
* denotes plane 1 of CNS-11643-1986.
! denotes plane 2 of CNS-11643-1986.

DEC Kanji. The DEC Kanji (OpenVMS/Japanese) code set currently supports ASCII, JIS X 0208-1983, and an area for UDCs, as shown in Table 1. The UDC area is further divided into the two subareas described in Table 2.

Table 2 The DEC Kanji UDC Area

AREA USAGE	QUADRANT	ROWS	NUMBER OF CHARACTERS	CODE RANGE
User Area	(1,0)	1-31	2,914	0xA121-0xBF7E
DEC Reserved	(1,0)	32-94	5,922	0xC021-0xFE7E

Recently, Super DEC Kanji, a revision and extension to the DEC

Kanji code set, has been proposed to support additional character sets such as JIS X 0201-1978 and JIS X 0212-1990, which are specified as follows:

CODE RANGE	ADDITIONAL PLANES
(SS2 1xxxxxxx)	JIS X 0201
(SS3 1xxxxxxx 1xxxxxxx)	JIS X 0212

The redefined UDC area includes both a user/vendor-defined area (UDA) and a user-definable character set (UDCS), as described in Table 3.

Table 3 The Super DEC Kanji UDC Area

AREA USAGE	QUADRANT	ROWS	NUMBER OF CHARACTERS	CODE RANGE
JIS X 0208 UDA	(1,1)	85-94	940	0xF5A1-0xFEFE
JIS X 0212 UDA	SS3 (1,1)	78-94	1,598	(SS3 + 0xEEA1)-0xFEFE
UDCS	(1,0)	1-94	8,836	0xA121-0xFE7E

DEC Hanzi. The DEC Hanzi (OpenVMS/Hanzi for Simplified Chinese) code set supports ASCII, GB 2312-80, and a UDC area described in Table 4.

Table 4 The DEC Hanzi UDC Area

AREA USAGE	QUADRANT	ROWS	NUMBER OF CHARACTERS	CODE RANGE
DEC Reserved	(1,1)	88-94	658	0xA1A1-0xFEFE
User Area	(1,0)	1-87	8,178	0xA121-0xF77E
DEC Area	(1,0)	88-94	658	0xF821-0xFE7E

DEC Hanyu. The DEC Hanyu (OpenVMS/Hanyu for Traditional Chinese) code set currently supports ASCII, CNS 11643-1986 (first and second planes), and the Digital Taiwan Supplemental Character Set (DTSCS). The DTSCS supplements the characters defined in CNS 11643-1986 with an additional collection of characters that address customer needs. Currently, the DTSCS defines the 6,319 characters recommended by the Electronic Data Processing Center (EDPC) of the Executive Yuan, a Taiwanese government body. The CNS 11643-1992 standard includes the DTSCS.

To support the additional DTSCS, the mixed 1-byte/2-byte scheme is extended to a 1-byte/2-byte/4-byte scheme. Each DTSCS character maps to a 4-byte code, in which a fixed leading 2-byte code (0xC2CB) combines with the following 2-byte code to form a 4-byte code. Of course, the code point 0xC2CB is reserved for this purpose. This scheme makes available another two 94 by 94 planes of code positions:

(0xC2CB 1xxxxxxx 1xxxxxx) (0xC2CB 1xxxxxxx 0xxxxxx).

Table 5 shows the current definition of the DTSCS. An additional area is available for UDCs in the CNS planes, as defined in Table 6.

Table 5 The DEC Hanyu DTSCS Area

AREA USAGE	QUADRANT	ROWS	NUMBER OF CHARACTERS	CODE RANGE
EDPC Recommended Characters	C2CB (1,1)	1-68	6,319	0xC2CBA1A1-0xC2CBE4B5
Reserved	C2CB (1,1)	68-94	2,517	0xC2CBBEB6-0xC2CBFEFE
Reserved	C2CB (1,0)	1-94	8,836	0xC2CBA121-0xC2CBFE7E

Table 6 The DEC Hanyu UDC Area

AREA USAGE	QUADRANT	ROWS	NUMBER OF CHARACTERS	CODE RANGE
UDC	(1,1)	93-94	145	0xFDCC-0xFEFE
UDC	(1,0)	82-94	1,186	0xF245-0xFE7E

DEC Hangul. The DEC Hangul (OpenVMS/Hangul) code set supports ASCII and KS C 5601-1987 (with the exception of UDCs).

Asian Text Input

The most widely used computer input device remains the keyboard. Because it is impossible to assign thousands of ideographic characters to a standard QWERTY keyboard, new methods must be devised to facilitate the Asian text input process. In this context, an input method is basically an algorithm that takes keystroke input representing certain attributes (e.g., phonetics) of a character or string and uses a lookup table to find characters or strings that have those attribute values. Typically, a user inputs several keystrokes and selects the desired character or string from a candidate list by means of an iterative dialog with the input method. This process is sometimes referred to as preediting. Depending on the physical location of where the dialog takes place, a preediting user interface can be one of three styles: off-the-spot, over-the-spot, or on-the-spot. Different input methods may have different preedit interface requirements. Usually, several screen areas are needed for the preediting dialog to take place. Input methods differ from culture to culture and from script to script.

The major difference in the implementation of input method support among the Asian OpenVMS variants is in the character cell terminal environment. Some input methods are suitable for programming into the terminal firmware. The Chinese and Korean input methods supported on the OpenVMS/Hanzi, OpenVMS/Hanyu, and OpenVMS/Hangul systems are examples of such methods. Other input methods are too complex or require so many resources as to make them too costly for firmware implementation. This is true of the Japanese input method, which needs to be implemented on the host. Such implementation causes a number of technical issues with the traditional ASCII character cell terminal-oriented application programming model, where an application does not have to be concerned with input methods and expects to receive character codes directly. The following three alternatives have been used to implement host-based input methods on the OpenVMS/Japanese system:

1. Application. All Japanese applications directly program the input method themselves. An application must call low-level routines (a set of kana-kanji input method routines are available in the JSY Run-time Library) to access the input method dictionary and directly controls the preedit interface in relation to its own screen management. This method is used by applications such as text editors, which need to directly manage the screen display. The method requires substantial reengineering of an ASCII application to support a Japanese input method.

2. Run-time library (RTL). Japanese applications call special text line input routines, which handle the Japanese input method. This method is suitable for applications that require simple line input of text. The RTL method hides the details of the input method from the application but lacks the flexibility to control the preedit user interface. The reengineering needed to handle the Japanese input method is shifted from the application to the RTL routines. This approach requires less application reengineering, but all standard line input routine calls in the application must be replaced by Japanese line input routine calls.

3. Front-end input processor (FIP). The Japanese input method is embedded as a front-end process inside the terminal queued I/O (QIO) system service. This method of implementation benefits existing high-level RTL text line input routines and requires little application or RTL reengineering to support the Japanese input method in the single-line input of Japanese text.[12]

The Asian OpenVMS graphical user interface on workstations is called Asian DECwindows/Motif. Current input method support is provided through a Digital extension implemented as an X client. With release 5 of the X11 standard, the implementation will migrate to using the standard X input method (XIM) support in the Xlib library routines.

Most Asian PCs have a front-end processor implementation of input methods resident on the PC. Therefore, PC desktop computers can send Asian characters directly when communicating with an Asian OpenVMS host.

The following is an overview of the input methods supported by each Asian OpenVMS variant.

Japanese Input Method. Kana characters can be typed directly on a standard keyboard using a kana keyboard layout. For kanji characters, the de facto standard input method is called the romaji/kana-to-kanji conversion, which is based on phonetic conversion. The process of entering a kanji string involves typing the kana (hiragana or katakana) or the romaji pronunciation of the string. The input method then looks in a conversion dictionary for the list of kanji strings that have the same pronunciation. Since most Japanese words have homonyms, the user usually needs to go through a selection process to find the desired kanji string. More advanced implementations involve performing syntactic and semantic analysis of the sentence to increase the efficiency of the input method. On the OpenVMS/Japanese system, the kana-to-kanji input method has a provision for separating conversion units into word, clause, and sentence. The method also has a learning capability that reorders the candidate list entries by means of a personal dictionary, putting the characters selected at the top of the list so that more frequently used words appear first in the homonym list.

Chinese Input Method. No standard exists for the Chinese input method. The large number of input methods that have been proposed over the years can be classified into one of two major types:

1. Pattern decomposition-based method. Each character is decomposed into basic strokes or patterns. Each stroke or pattern, e.g., a root radical, is assigned a code (mapped to a key) and each character is retrieved by inputting a sequence of such codes.

2. Phonetic-based method. Each character is transcribed into phonemic letters and retrieved by this phonemic transcription. The system used in Taiwan is based on the National Phonetic Alphabet (Bopomofo), whereas the PRC uses Roman alphabets based on the Wade-Giles system.

The OpenVMS/Hanzi system supports the following Chinese input methods:

- Five stroke

- Five shape

- Pinyin

- Telex code

- GB 2312 code

The OpenVMS/Hanyu system supports the following Chinese input methods, which are implemented by firmware on the Digital VT382 series Chinese terminals:

- Tsang-Chi

- Quick Tsang-Chi

- Phonetic

- Internal code

- Phrase

Korean Input Method. Hangul characters are composed by directly typing the individual Hangul letters. The composition sequence always starts with a consonant, is followed by a vowel, and finishes with a consonant, if present. The input method validates the composition sequence keyed in by the user at each step. The display device updates the intermediate rendering of the partially formed Hangul character as the shape and position of each letter changes during composition. Hanja characters are entered by typing their Hangul pronunciation. The input method displays a list of all possible Hanja characters (homonyms). More sophisticated implementations can perform Hangul-to-Hanja conversion in word units similar to that of the kana-to-kanji conversion. On the Digital VT382 Korean terminal, both the Hangul and the Hanja input methods are implemented by firmware.

Asian Text Output

Asian character fonts are usually displayed or printed as bit-map graphics. To meet the requirements of specific applications such as scaling and plotting, these fonts can also be defined as outline fonts using vector representation. International codes of Asian language characters are mapped to the corresponding font data when needed for output. Predefined character fonts are usually stored in the read-only memory (ROM) of terminals and printers for better performance. As for the English alphabet, different standards, styles, and sizes exist for Asian language character fonts. The following list contains some of the more popular font styles used in the respective markets:

MARKET	FONT STYLE
Japan	Mincho, Gothic, Round-Gothic
Korea	Myuncho, Gothic
PRC	Song, Quasi-Song, Hei (boldface), Kai
Taiwan	Sung, Hei (boldface), Kai

In general, Asian ideographic characters require high-definition fonts, i.e., at least a 24-by-24 dot matrix, to achieve acceptable visual quality. As a result, memory requirement is a major issue when supporting Asian fonts.

Hardware

Supporting Asian language processing requires modifying the standard video terminals and printers. In general, software products need to recognize the different functional characteristics of Asian terminals and printers. For example, the character set designation and invocation defaults differ from those of standard terminals.

Workstations do not require any modifications (except for exchanging a local language keyboard for the standard one), because input and display are directly supported by software.

Asian Video Terminals. The traditional character cell terminal provides certain local display and input functions on behalf of a software program. For example, the terminal firmware preprocesses scan codes generated by keyboard input and converts them to character code before sending them to an application. Similarly, character fonts are usually stored in the terminal ROM. Digital has developed a variety of video terminals to support Asian language processing.

Some major hardware considerations for Asian video terminals are

- o High-resolution video display. Ideographic characters have complex glyphs, which require at least a 24-by-24 dot matrix cell to be of acceptable display quality. Such a cell would occupy two ASCII columns. As a result, to maintain a 26-line (40 ideograms per line) display requires a screen resolution of at least 960 by 780 pixels. Typically, Digital's Asian video terminals use monitors that run at a 60-hertz noninterlaced mode, a mode substantially higher than that of standard ASCII terminal monitors.

- Font memories and loading protocol. The terminal requires additional ROM to hold the fonts of standard characters in an Asian character set, typically 7,000 to 20,000 characters. Also, for characters outside the standard set, i.e., UDCs, the terminal requires random-access memory (RAM) to downline load the fonts from the host. Digital Asian terminals support font-loading protocols that work with the host software to downline load fonts into RAM either on demand or on a preloading basis. The font cache in Digital's Asian terminals can usually hold about 400 characters at once.

- Input method. Implementing input methods on a video terminal requires additional hardware modification. The input method algorithms must be programmed into the firmware together with extra memory for the input method lookup tables. In addition to the main display area, one extra line on the screen is needed as an input method work area, e.g., for displaying candidate lists for user selection. Some keys must be assigned permanently for invoking different input methods. The printing of legends on the tops of the keys is now more complex, because the keytops must include additional legends for the input method keyboard layout. For example, on Digital's Hanzi terminals, four ideograms must appear on the tiny area of one keytop.

Asian Printers. Digital supports a range of Asian printers. Similar to Asian video terminals, Asian printers must support font-loading protocols to downline load fonts for UDCs by either preloading or on-demand-loading methods. Additional RAM is required to hold these fonts. Also, Digital's Asian printers generally support multiple font typefaces and sizes.

Asian OpenVMS Structure

The components provided by the Asian OpenVMS variants on top of the standard OpenVMS system can be divided into five main groups:

1. System support for transparent processing of UDCs

2. An enhanced OpenVMS terminal I/O subsystem to support Asian terminal devices

3. A set of run-time libraries to facilitate Asian application development on Asian OpenVMS systems

4. A set of localized utilities and commands for users to perform common tasks on OpenVMS systems in their native languages

5. A utility to set the operating modes (standard OpenVMS mode or Asian OpenVMS mode) of the localized components

Figure 4 summarizes the Asian OpenVMS system structure.

Figure 4 Asian OpenVMS System Structure

```
+-----------++--------------+ +---------------++-----------+
|           || LOCALIZED    | |               ||           |
|           || OPENVMS      | |  USER         ||           |
|           || COMMANDS AND | |  APPLICATION  ||           |
|           || UTILITIES    | |               ||           |
| USER-     |+--------------+ +---------------+| MODE      |
| DEFINED   |+--------------------------------+| SWITCHING |
| CHARACTER || MULTIBYTE PROCESSING RTL       ||           |
| SUPPORT   || LOCALIZED SCREEN MANAGEMENT RTL||           |
|           || LOCALIZED OPENVMS CALLABLE     ||           |
|           ||   UTILITY ROUTINES             ||           |
|           |+--------------------------------+|           |
|           |+--------------------------------+|           |
|           ||  ASIAN TERMINAL I/O SUBSYSTEM  ||           |
|           ||                                ||           |
+-----------++--------------------------------++-----------+
```

ASIAN OpenVMS COMPONENTS

This section reviews the major components of the Asian OpenVMS variants.

User-defined Character Support -- The Character Manager

Attributes of characters in the standard character sets supported on an Asian OpenVMS system are known and fixed. Therefore, attribute support can be built into the system statically. In contrast, UDCs usually require their attributes to be dynamically defined and accessed. A new utility called the character manager (CMGR) enables users to create, manage (modify and update), and retrieve UDCs and their attributes. UDC support is currently offered on the OpenVMS/Japanese, OpenVMS/Hanzi, and OpenVMS/Hanyu systems. In the OpenVMS/Hanyu system, the CMGR also supports Digital-defined characters, e.g., the DTSCS and DEC Recommended Characters (DRC).

The CMGR manages a set of systemwide databases that store UDC attributes. Two UDC attributes are currently supported, glyph images and collating values.

To represent the UDCs in the computer, the CMGR allows a user to assign each UDC a code point in the designated UDC area.

Currently, UDC characters are entered by directly typing their binary code. The code point serves as the key in the CMGR databases for retrieving other attributes of the character.

The CMGR utility provides a user interface to create and manage the UDC attribute database. The user interface includes a font editor for users to create the glyph image of a UDC and entries for other attributes. To allow applications to retrieve the UDC attributes, the CMGR has a set of application programming interfaces (APIs) used to access the individual attribute databases. In particular, the on-demand font loading of UDCs supported by the Asian terminal I/O subsystem employs the CMGR font databases, and the SORT/MERGE utility uses the collation databases for UDC sorting.

CMGR Font Database. To output a UDC to a display or printing device, the UDC's glyph image must first be defined. The CMGR provides a screen font editor for users to create the glyph images. The CMGR supports multiple typefaces (e.g., Hei, Sug, and Default) and font sizes (e.g., 24 by 24, 32 by 32, and 40 by 40) in multiple databases. There are two ways to load the UDC fonts to Asian output devices, namely, preloading and on-demand loading.

Fonts can be preloaded by sending a file that contains the appropriate control sequences and font patterns, which are discussed in more detail later in this section. The CMGR provides a command that generates a preload file from the font database for required UDCs.

On-demand font loading is a more complicated mechanism, which involves an on-demand loading protocol. Font patterns are retrieved from the font database through the CMGR callable interface by a font-handling process.

CMGR Collation Attribute Database. To facilitate the sorting of data, including UDCs, the collation weights of the characters must be defined. Currently, only the OpenVMS/Hanzi and OpenVMS/Hanyu systems offer this feature.

Asian Terminal I/O Subsystem

The Asian terminal I/O subsystem is an extension of the standard OpenVMS terminal I/O subsystem. It consists mainly of the OpenVMS terminal class drivers/port drivers, auxiliary class drivers, and server processes, and handles both standard and Asian terminals simultaneously. For Asian terminals, the subsystem provides extended functions to support multibyte character handling in the terminal QIO system service, input method, code set conversion, and font loading.

Terminal QIO System Service/Multibyte Character Handling. The enhanced terminal QIO system service can handle mixed ASCII and multibyte Asian characters in line input calls. Line editing (e.g., character echo, cursor movement, character deletion, character insertion, word delimiters, and character overstrike), line wrapping, uppercasing, and read verifying will handle Asian characters correctly. Because the QIO system service is the lowest-level routine that handles terminal I/O, all other text I/O routines such as LIB$GET_INPUT, $GET RMS service, and the text I/O facility of programming languages such as C, Fortran, and COBOL are layered on it. The enhancements automatically benefit all of these higher-level routines.

Font Loading. Asian terminal devices have writable font memory (WFM), and the firmware supports font-loading sequences and logic. A text file is scanned by a utility program prior to output to a terminal or printer. The Asian terminal I/O subsystem then creates a preloading file, which contains the font-loading sequence for all nonresident characters found in the file. Next, the subsystem sends this preloading file to the terminal or printer, causing the required fonts to be loaded in the font memory. Finally, the text file is output to the terminal or printer. This method is limited by the size of the font memory, typically 300 to 500 characters. The font preloading method is used mainly in batch operations, such as line printers, where performance is an important factor.

When an Asian video terminal or printer receives an Asian character code and determines that it is a UDC, the terminal firmware automatically halts the current processing and generates a font request to the OpenVMS system. The terminal driver traps this request and passes it on to a process called the font handler. On behalf of the terminal, the font handler retrieves the font bit map of the requested character from the system font database and sends it back to the terminal or printer, which in turn loads it into its RAM and resumes the display processing. Because it involves XON/XOFF flow control, which is done at a very low level of the system, the process requires modifications to device drivers. The amount of UDC font is not limited by WFM capacity, because the terminal firmware automatically updates the memory.

Front-end Input Process (FIP).[12] One of the biggest differences between Japanese and other Asian language (e.g., Chinese and Korean) support on the OpenVMS system is in the implementation of the input method. The nature of the kana-to-kanji input method makes it unsuitable for implementation in terminal firmware. The method requires a huge input method dictionary (about 1 megabyte in size) and a dynamic memory work area for syntactic and semantic analysis. Also, updating an input method dictionary that is implemented in firmware is a very costly operation.

Code Set Conversion. Prior to the introduction of the Asian OpenVMS variants, Digital's customers used video terminals and printers that support proprietary local language code sets from third-party vendors. To protect customer investments and to ensure a smooth migration path for legacy equipment, the Asian terminal I/O subsystem provides an application-transparent, code set conversion facility. This facility is based on the terminal fallback facility (TFF) introduced in OpenVMS version 5.0, which provides a similar function for conversion between 7-bit National Replacement Character Sets (NRCSs) and the 8-bit DEC MCS. TFF provides a middriver that converts both incoming and outgoing data from one code set to another. For the Asian OpenVMS variants, the conversion logic is extended to support 16-bit character entities. Currently, TFF supports the conversion between the DOOSAN code and the DEC Hangul code on the OpenVMS/Hangul system and the MITAC TELEX code and the DEC Hanyu code on the OpenVMS/Hanyu system.

In addition, code set conversion is necessary between heterogeneous systems because of the proliferation of encoding schemes used by different vendors. For instance, Chinese PCs in Taiwan use the BIG 5 code. To facilitate the communication between the OpenVMS system and PC desktop computers, the OpenVMS/Hanyu system supports the conversion between the BIG 5 code and the DEC Hanyu code.

Asian Application Programming Support

To help software developers write Asian applications on Asian OpenVMS variants, Digital provides a set of common Asian multibyte character processing RTL routines to supplement the standard OpenVMS RTLs. In particular, our Asian localization effort to develop OpenVMS layered products utilizes these RTLs. Functions provided by the Asian language RTL (approximately 240 routines) are classified into the following categories of routines:

- Character conversion
- String
- Read/write
- Pointer
- Comparison
- Search
- Count

- Character type

- Date/time

- Code set conversion

The majority of the routine interfaces are common to all Asian countries. Currently, one library image supports the Hanzi, Hanyu, and Hangul language variants. Language-specific code is hidden under this generic multibyte interface and switched at run time by a system logical name defined during system start-up.

The OpenVMS/Japanese system has a set of routines for handling kana-to-kanji conversion, both high level and low level. The high-level routines, such as JLBGET_INPUT, JLBGET_COMMAND, and JLB$GET_SCREEN (Japanese versions of LIBGET_INPUT, LIBGET_COMMAND, and LIB$GET_SCREEN), hide the kana-to-kanji input method details from the application. These routines use the off-the-spot preediting that usually takes place at the last line of the screen; however, the flexibility of the preedit user interface is limited. A set of low-level routines performs primitive functions such as opening the conversion dictionary, finding the next candidate kanji string, and getting the contents of the internal buffer. The kana-to-kanji input method is programmed by calling a sequence of these routines. This implementation gives the application the ability to directly control the screen management and allows flexibility in the design of the preedit user interface; however, the application must deal with every detail of the input method, which is a disadvantage. In addition, the library IMLIB helps the application customize the keyboard mapping for kana-to-kanji conversion.[12]

The screen management (SMG) RTL on the OpenVMS system provides a suite of routines for designing, composing, and keeping track of complex images on a character cell video terminal in a device-independent manner. The standard SMG version supports only the ASCII and DEC Special Graphics character sets and cannot correctly handle multibyte Asian characters. For example, operations such as screen update optimization, boundary processing (clipping on borders), and cursor movements operate on part of a multibyte Asian character and cause screen corruption because of the "one-character-is-equal-to-one-byte" assumption.

The Asian OpenVMS variants provide an extended version of SMG (about 20 percent of the original routines have been extended) to support multibyte character sets and DEC MCS, in addition to ASCII and DEC Special Graphics. To maintain downward compatibility, most routine entries remain identical, with an optional character set argument added at the end of the argument list to indicate desired character set operations. Alternatively, users can define a logical name SMG$DEFAULT_CHARACTER_SET without explicitly passing the character set argument in the routine call. Existing ASCII applications run unmodified with the Asian SMG. New Asian applications that use multibyte features relink with the new library.

Asian Commands and Utilities

The OpenVMS user interface determines the way an end user interacts with the system. The interface includes such components as the DCL command line interpreter, system help and messages, and all the system utilities provided by the OpenVMS system. Selected user interface components of the OpenVMS system have been localized to support Asian characters on the Asian OpenVMS variants. A description of some of these localized components follows.

DCL Command Line Interpreter. The algorithms in the standard DCL that assume characters to be equal to 1 byte and interpret these characters as ASCII/DEC MCS are enhanced for the following DCL primitives in the Asian code set modes:

- Command parsing. Parsing of command input in single-byte units causes data corruption, because part of some multibyte Asian characters can be mistaken for one of the special DCL ASCII characters such as !, @, or ". Command parsing is now done in character units instead of byte units, and operations such as terminator, delimiter checks, and quotation mark compression are skipped on Asian characters, since the DCL special characters are all in ASCII.

- Character uppercasing and lowercasing. Uppercasing and lowercasing are applied only to ASCII characters, because the concept of uppercase/lowercase does not exist in Asian character sets. Uppercasing/lowercasing in single-byte units corrupts Asian character data, because part of an Asian character can be indiscriminately uppercased/lowercased.

- Symbols and labels. Certain 8-bit values (those with no character assigned in the DEC MCS) are currently disallowed for DCL symbol names, symbol values, and labels. This restriction has been removed in the Asian modes to allow all Asian characters in DCL symbols and labels. The enhanced algorithms maintain separate symbol tables for each of the code set modes, because of the possibility of code collision issues across different code sets.

The Asian DCL command line interpreter is currently supplied with the OpenVMS/Hanzi, OpenVMS/Hangul, OpenVMS/Hanyu, and OpenVMS/Thai systems in the same binary image, i.e., a single image supports multiple code sets. The default code set mode for

DCL for a particular system is established during system start-up by means of a defined logical name supplied with the start-up procedure of each Asian OpenVMS variant. Switching the code set mode between DEC MCS and the particular Asian code set of the system is accomplished through a utility, e.g., HANZIGEN in the OpenVMS/Hanzi system. The Asian DCL is not supplied with the OpenVMS/Japanese system, because until only recently the Japanese input method was not available at the DCL level.

System Help and Messages. The OpenVMS/Hanzi, OpenVMS/Hanyu, and OpenVMS/Hangul systems include a translated Asian language version of the OpenVMS system help library (accessed by typing HELP at the $ prompt). The Asian version of the system help library is placed in a directory that is separate from the original English one but that has the same file name. The user can switch the language (English or the particular Asian language) of system help by using the ASIANGEN utility, which redefines the file specification logical to point to the appropriate file.

The OpenVMS/Japanese system provides a translated Japanese version of the system messages (SYSMSG.EXE), which is placed in a subdirectory of SYS$MESSAGE. Users can switch the language of the system messages by using the SET LANGUAGE command, which reloads the message file into memory.

In addition, most of the localized original utilities and Asian-specific utilities provide bilingual help and messages.

SORT/MERGE. Collation rules in the Asian languages are very different from those of the Latin languages.[13]

- o   Asian collation sequences. An Asian character has different collation sequences based on different attributes. The SORT/MERGE command is extended as follows to include new subqualifiers for the Asian collating sequences: /KEY=(POS:m, CSIZE:n, <collating sequence subqualifier>). The Asian OpenVMS SORT/MERGE utility supports the Asian collating sequences shown in Table 7.

- o   Collation weights. Unlike ASCII, the collation weights of the Asian collating sequences cannot be derived by virtue of the code value. Rather, the string comparison for Asian collation sequences are driven by collation weight tables. For the standard characters, these tables are built into binary images that are linked with the utility for fast access.

- o   Multibyte characters. String comparison in the original SORT/MERGE operation is done in byte units, because a character is assumed to be equal to 1 byte. For the Asian SORT/MERGE, a comparison operation must be aligned by character, i.e., multibyte, units rather than by byte units. The operation must be able to handle the case in which the start position of a sort key (specified by a byte position) in a record is in the middle of a multibyte character. Also, to avoid a truncation problem at the key boundary, the size of the sort key (mixed ASCII and multibyte characters are allowed) is specified as a number of characters instead of a number of bytes.

- Multiple passes. Sorting Asian characters by any of the individual collating sequences (except QuWei) may not produce a unique sort order. In general, multiple successive passes using different collating sequences are needed to do so. Thus, the Asian OpenVMS SORT/MERGE utility allows a sort key specified with multiple passes of different collating sequences. In addition, if the /STABLE qualifier is not specified, QuWei collation is always added last to the sort key to further classify records with identical collation values.

- User-defined characters. The Asian OpenVMS SORT/MERGE utility supports collation of UDCs. When a UDC is encountered, the SORT/MERGE operation retrieves the collation weight from a system database maintained by the CMGR utility with the value defined by a user when the character was registered.

Table 7 Asian Collating Sequences Supported by the OpenVMS User Interface

COLLATION SEQUENCE TYPE	OpenVMS/JAPANESE	OpenVMS/HANZI	OpenVMS/HANYU
Pronunciation	Onyomi* Kunyomi! Kokugo!! Kana8bit	Pinyin	Phonetic_Code
Radical	Bushu	Radical	Radical
Stroke Count	Sokaku	Stroke	Stroke
Internal Code	JIScode	QuWei	QuWei

Notes:
* denotes a Chinese reading.
! denotes a Japanese reading.
!! denotes a Kana reading.

MAIL. Most of the work involved in localizing the MAIL utility enhances the user interface to use Asian characters. String search enhancements allow processing by character units instead of by byte units. String uppercasing is not applied to Asian characters. The subject field, the personal name field, and the folder names can all contain Asian characters. The listing of mail folders can be displayed in sorted order in any of the supported collation sequences using the new command qualifier DIR/FOLD/COLLATING_SEQUENCE=(<collating sequences>).

The MAIL utility invokes the Asian text editors by default instead of invoking the standard ones. The OpenVMS/Japanese system incorporates the Japanese input method to allow users to enter Japanese characters.

EDT. The Asian OpenVMS EDT editor was localized and enhanced for Asian text editing. Much of the work involved driving the terminal display correctly for Asian characters. In addition, the editor has a large number of new editing features.

TPU/EVE. Localization of TPU and EVE deals mainly with managing the screen update for mixed ASCII and Asian characters, such as cursor movement and screen boundary handling. Both the TPU editing engine and the EVE interface were modified. Asian-specific TPU built-in procedures were added, and existing ones were enhanced. String search is now aligned at the character boundary rather that on byte units.

For the Japanese TPU/EVE, one of the most difficult tasks is to incorporate the Japanese input method. This requires managing overlap windows in a character cell terminal between the input method working area and the background editing area.

DECwindows System. With the increasing emphasis on internationalization features in the X11 and OSF/Motif standards, OpenVMS DECwindows systems provide these features and the localization features demanded by the market. For a description of the latest internationalization support in the X Window System standard, refer to the book by Scheifler and Gettys.[14]

ASIAN OpenVMS LOCALIZATION ISSUES

The Asian OpenVMS effort has been addressing various technical and engineering issues.[15,16,17] This section discusses the major ones.

Technical Issues

Localization of the OpenVMS components to support the Asian languages requires reengineering the program codes and text translation. The need to reengineer source code arises for two main reasons.

1. OpenVMS components make fundamental programming assumptions and practices based on the ASCII and DEC MCS character sets. For example,

    - OpenVMS components assume the character set to be

ASCII (plus DEC MCS in some cases), and blindly uppercase and lowercase characters, validate characters against the DEC MCS, and define printable characters according to the ASCII and DEC MCS encodings.

- OpenVMS components assume characters to be 1 byte and use string manipulation algorithms based on 1-byte units.

- OpenVMS components assume the display width of a character to be of fixed length (1 byte) and use screen display management algorithms based on the assumption that 1 byte equals one display column.

- OpenVMS components assume that the character count, the byte count, and the display width are the same, and use string manipulation algorithms and character cell terminal screen display management based on this assumption.

2. Some functionality that is required to support Asian languages is missing in the standard OpenVMS environment. For example,

   - Keyboard input of Asian characters requires more complicated input method processing than is available in the standard OpenVMS environment.

   - Collation rules of Asian languages are radically different from English collation rules, on which the standard OpenVMS environment is based.

   - The standard OpenVMS environment does not support the application-transparent processing of UDCs.

   - The writing direction of Asian languages can be vertical, i.e., from top to bottom. The standard OpenVMS environment assumes horizontal, left-to-right languages.

Engineering Issues

Historically, the Asian localization of the OpenVMS system has been organized as an engineering effort that is separate from mainstream development. As a result, a number of engineering constraints and overhead costs exist.

- Single language support. The design goal for the Asian OpenVMS variants, as driven by the local market requirements, has been targeted at supporting a single language on one system, i.e., one language variant per system. As a result, no special design considerations are given to supporting multiple languages on one system.

- Full upward compatibility. The top design requirement is to keep full downward compatibility with original ASCII/DEC MCS OpenVMS systems. All ASCII/DEC MCS applications with existing data must be able to run unchanged on the Asian OpenVMS variants. In fact, an Asian OpenVMS system can, at any time, be reset to operate in the original DEC MCS mode, if desired. Therefore, most localized components must be able to switch between the standard and Asian code paths. System mechanisms for determining the current language variant and operating mode are required.

- Optimal performance. Another design goal is to minimize any performance impact on standard English components. As a result, Asian codes are designed around standard code paths. For example, branches for Asian code are placed at the end of a conditional statement, and Asian code branches out from the main line code using special hooks.

- Limited or no kernel changes. Since Asian code changes are not merged into the mainstream, kernel changes in Asian code would be very difficult to maintain with new OpenVMS releases. In addition, any kernel changes in the standard OpenVMS release will likely break the Asian code. This puts a constraint on supporting Asian languages in OpenVMS kernel components.

- Commonality. Because the Asian languages share a lot of commonality, techniques such as common source are used for most Asian localized components to maximize engineering return by sharing common Asian localization code.

CONCLUSIONS

Local language processing has become a mandatory functionality for computer systems sold in Asian markets. From the OpenVMS operating system perspective, the basic local Asian language processing requirements are being addressed by its Asian language variants in a single-language-for-a-single-locale manner. With global trade and the technology trend of distributed computing systems, the challenge for the future is to be able to provide OpenVMS services simultaneously to multiple clients operating in different languages and code sets. Such a requirement leads to the concept of a multilingual operating system, which allows software applications to run irrespective of the language and/or code set they support. With the availability of the ISO 10646 Universal Character Set (UCS) standard, the set of tools for building such a multilingual operating system has been enhanced.[18]

From an engineering perspective, the current Asian localization approach of OpenVMS has been adopted historically because of a number of factors and constraints, such as the organization of engineering resources and the initial need to bring the capability rapidly to the market. The reengineering techniques are geared toward the character set encoding schemes currently supported. The arrangement of performing localization remotely and independently from the original mainstream development has meant costly reengineering and maintenance overheads in the long term. With the industrial trend of shipping global software simultaneously satisfying multiple different local market requirements, an international product engineering approach must be taken to minimize the cost of worldwide system engineering to deliver a global product. In particular, the original product must be internationalized from the ground up, so that no separate reengineering is needed during localization to support a local market. In addition, to achieve simultaneous worldwide delivery, concurrent engineering of localization needs to be performed closely in parallel with the product development.

REFERENCES

1. T. Greenwood, "International Cultural Differences in Software," Digital Technical Journal, vol. 5, no. 3 (Summer 1993): 8-20.

2. Code of the Japanese Graphic Character Set for Information Interchange, JIS C 6226-1978 (Tokyo: Japanese Standards Association, 1978).

3. Code of the Japanese Graphic Character Set for Information Interchange, JIS X 0208-1983 (Tokyo: Japanese Standards Association, 1983).

4. Code of the Japanese Graphic Character Set for Information Interchange, JIS X 0208-1990 (Tokyo: Japanese Standards Association, 1990).

5. Code of the Supplementary Japanese Graphic Character Set for Information Interchange, JIS X 0212-1990 (Tokyo: Japanese Standards Association, 1990).

6. Code for Information Interchange, JIS X 0201-1976 (Tokyo: Japanese Standards Association, 1976).

7. Code of Chinese Graphic Character Set for Information Interchange, GB 2312-1980 (Beijing: Technical Standards Publishing, 1981).

8. Standard Interchange Code for Generally-used Chinese Characters, CNS 11643-1986 (Taipei: National Bureau of Standards, 1986).

9. Chinese Standard Interchange Code, CNS 11643-1992 (Taipei: National Bureau of Standards, 1992).

10. Code for Information Interchange (Hangul and Hanja), KS C 5601-1987 (Seoul: Korean Industrial Standards Association, 1989).

11. Information Processing -- ISO 7-bit and 8-bit Coded Character Sets -- Code Extension Techniques, 3d ed., ISO 2022 (Geneva: International Organization for Standardization/International Electrotechnical Commission, 1986).

12. T. Honma, H. Baba, and K. Takizawa, "Japanese Input Method Independent of Applications," Digital Technical Journal, vol. 5, no. 3 (Summer 1993): 97-107.

13. R. Haentjens, "The Ordering of Universal Character Strings," Digital Technical Journal, vol. 5, no. 3 (Summer 1993): 43-52.

14. R. Scheifler and J. Gettys, X Window System, X11, Release 5, 3d ed. (Burlington, MA: Digital Press, Order No. EY-J802E-DP-EEB, 1992).

15. Introduction to Asian Language Software Localization (Maynard, MA: Digital Equipment Corporation, Order No. AD-PG0AA-TE, December 1990).

16. Technical Guide to Asian Language Software Localization (Maynard, MA: Digital Equipment Corporation, Order No. AD-PG0BA-TE, December 1990).

17. Addendum to Technical Guide to Asian Language Software Localization (Maynard, MA: Digital Equipment Corporation, Order No. AD-PG0CA-TE, December 1990).

18. Information Technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane, ISO/IEC 10646-1 (Geneva: International Organization for Standardization/International Electrotechnical Commission, 1993).

TRADEMARKS

BIOGRAPHY

Michael M. T. Yau   Michael Yau is a principal software engineer in the International Systems Engineering Group. Since 1984, he has worked on Asian language support in the OpenVMS operating system. He led and managed the development team in Hong Kong from 1986 to 1991. Currently, he provides architecture and product internationalization support to U.S. engineering groups. Prior to joining Digital, Michael worked for GEC Marconi Avionics (U.K.). Michael holds a B.Sc.(Hons) in mathematics and an M.Sc. in communication engineering from the Imperial College of Science and Technology, University of London.