Chapter 5

Japanization

There are several ways and levels of japanizing a software product. Some few products sell even without been japanized (mainly computer games and niche-market products). Japanese EDP specialists are used to work with English products. However the normal Japanese user is (or will not be) satisfied by using foreign language software products.

Even if it does not look so, the entry of personal computer systems in Japanese offices was not that long ago. About five or seven years ago the most used computer system in Japan was the mini to main frame range of computers. Since the fast development of personal computers started, the smaller systems got more powerful to handle the resource swallowing Japanese features (like Kanji support, ...).

5.1 First Steps

The first step of Japanization is to implement the program on a Japanese hardware platform. The problem with, e.g., the Japanese PC platforms is that they all have different hardware layouts and specifications compared to an IBM PC. Only " clean " programs could be transferred and (maybe) used as a plug and play version (e.g., for a demo) in a Japanese MS-DOS environment. The usual way is to recompile the whole program with a Japanese version of the compiler. Before that you have to change all hardware depended BIOS calls and get rid of " dirty " code. You have to do this because in Japan there are about seven different PC hardware platforms. This makes

it very difficult to localize a product which peeks and pokes around in the system. If you use, e.g., undocumented MS-DOS calls you can not be sure that this call exists in the Japanese version of a hardware vendor MS-DOS version.

If you have " cleaned up " your program and compiled it on a Japanese hardware platform you will have an original version of your program which runs on (one) Japanese hardware platform. This version has the big disadvantage that it does not support any of the Japanese language specialities.

If it is a special product, e.g., for system administrators or other EDP specialists you maybe could sell it to Japanese customers.

5.1.1 Adding Japanese features

Some big international companies, like Citibank, use their original programs and add a Japanese Front End (do not mix this up with Front End Processor). This means that the program body is not changed, it can not work with Japanese Kana or Kanji characters. The new Front End provides the Japanese user just with a Japanese user interface, like Japanese menus and help-screens. The functions of the program are not changed. This is a way to make it easier for Japanese computer user to use a program in an international environment. Nevertheless this approach applies only for big international companies or programs which not have to work with Japanese data.

The most important step in japanizing a software product is to enable it to work with DBCS characters, i.e., with the Japanese syllable alphabets Hirgana, Katakana and the ideographic Kanji characters. In order to do this you could create a Japanese, a Chinese, a Korean, ... version or you enable your product to run in every national language DBCS environment (more about that topic later).

If you want to enable your software to work with DBCS character sets, i.e., you have to use data-types which are able to handle this type of characters (like the wchar_t data-type in C or other specific Japanese data-types provided by the compiler maker).

After doing this there are two ways to choose from. You could launch the first version of your DBCS enabled program without changing the menus and help-screens, just by adding a Japanese manual. The second, more complicated, way is, not only to enable your product to handle Japanese characters, but also to adapt the manual, menu and help system to the Japanese language. The first way could be a good choice

if you just want to be present on the Japanese market and later on (if the product seems to be successful) you want to launch a " more " japanized version.

It is clearly visible that the version with DBCS support, japanized menus and help-screens will be more successful in the Japanese marketplace. For the beginning or for niche-market products a version with DBCS support and, e.g., English menus would also sell.

5.1.2 Full Japanization

The last stage in japanizing a software product is to adapt all cultural Japanese specialties which apply for this product. For a word processor this would be things like Amikake, Keisen, Kinsoku Shori, Rubi characters, Kinto-waritsuke and so on. If you japanize a real estate agent program the Japanese user would appreciate if the system would support the Japanese units for measuring the floorspace (e.g., Tsubo).

It is naturally that everybody likes to work in well know environment. Imagine a Japanese user who has to work with, e.g., a program which prints the date in the US date format (MM/DD/YY). This is just a minor difference but it could cause misunderstanding. In addition the user could not use the system to prepare papers for the government because they insist on the Japanese date format.

If you support all the other small cultural differences it will help your system to compete on the Japanese software market.

5.1.3 Japanese Operating Systems

To give the appropriate support for the Japanese user the operating system has to support several features. First of all the Japanese user will appreciate if he could use file, volume and device names written either in Hiragana, Katakana or Kanji characters. Furthermore a complete help system and error messages in Japanese should be available.

For applications, commands and programming languages it is necessary to provide the possibility to write comments and the names of variables in Japanese. In addition it should be possible to use special Japanese data-types (for DBCS characters). A nice feature would be that reserved words could be written in Japanese.

5.2 Japanese Software Environment

The PC/WKS market will become the most important market in Japan. The two leading operating systems are MS-DOS and UNIX. Both are existing in japanized versions. The PC market is a very diverse market with many different hardware platforms and implementations of MS-DOS. Since the MS-Windows market share in Japan is growing the difference between the operating systems has started to fade away.

The UNIX workstation market is more homogeneous. Even if there are different hardware platforms and UNIX implementations. The different UNIX operating systems implementations, which are always licensed by the UNIX Systems Laboratory, uses all the same program interface. So if you use a UNIX system there should be only minor problems in adapting a software to this environment (at least less then in the Japanese MS-DOS environment).

5.2.1 MS-DOS

The Japanese DOS market is a mess (!). There are several vendors which offer their hardware with MS-DOS. Unfortunately each platform has a different hardware layout and implementation of MS-DOS. The best example is the market leader NEC with the PC98XX series. This machine uses an architecture which is similar to the architecture of the IBM PC. Nevertheless the machines are incompatible because the engineers of NEC used, e.g., different hardware addresses for controller chips, placed other interrupt routines behind interrupts as IBM did and designed a more complex video memory design for the handling of Japanese characters. NEC is not the only vendor who did this. All of the other vendors (like Fujitsu, Toshiba, ...) did the same thing.

Moreover the mess of different implementations there are also many different FEPs for PCs available in Japan. If your program was developed by using, e.g., the VJE-β then you can not be sure that an other FEP supports the same features as this FEP. Switching or controlling the mode of the FEP is not possible except this feature are provided by the standard FEP interface.

Clean Programs

The only programs which should run on every MS-DOS machine are really " clean " programmed programs. This means that you only could use the fully documented MS-DOS calls. If you use undocumented DOS calls, BIOS calls, your system peeks & pokes around or you try to manipulated the hardware directly your system will probably not run on a Japanese MS-DOS machine. This has caused many obstacles for software developers in and outside Japan. For every hardware platform you have to code a special version or if you would use the standard MS-DOS interface your program would be slower or could not perform some special program features. This was and is the main hurdle for foreign software vendors.

Japanese MS-DOS

The good thing about the Japanese MS-DOS is that it is fully japanized and supports many the Japanese special cultural features. The help-system provides the help-screens in Japanese (see figure 5.3 on page 176). The system allows to use file names written in Japanese (see figure 5.4 on page 179). There are many successful business applications available in japanized versions (see the picture of LOTUS 123j in figure 5.4 on page 179). Even with the bunch of hurdles and problems the system is the best choice for running PC software on Japanese PC systems.

AX & OADG DOS

Fortunately there are two new versions of the PC operating system around which try to make it easier for the user and the software developers to run programs on different platforms. These new DOS versions are from the AX Consortium and from the OADG (Open Architecture Development Group, called DOS/V). Both DOS versions run on different platforms from different vendors. Both systems support special Japanese features like FEP, Kanji characters, ... The good thing about this new approach is that a program developed on an AX (or OADG) DOS machine *should* run on every other AX (or OADG) DOS machine. It is still not possible to peek and poke around or manipulate the hardware directly[1], but at least all the DOS calls follow one (AX

[1]You should not do it, but AX machines have all the same hardware architecture so it should be fairly possible to run a " dirty " programmed AX program on every AX compatible machine

or OADG) standard. The latest developments show that both systems will become compatible to each other in the near future.

Japanese Windows

The biggest boost for PC software compatibility, on different platforms, was the Japanese MS-Windows version 3.0 (see figure 5.3 on page 176). This version of MS-Windows was still implemented by each hardware vendor, but all implementations use the same set of windows APIs (Application Program Interface). In addition the Japanese window's version is compatible to the English (German, French, ...) window's version. The only problem with the compatibility between the different national versions is that national characters or line elements will be displayed as Katakana characters by the Japanese window's version.

This fact makes it easy to run foreign software in a Japanese computer environment. If the programmer has used the wide-character supporting windows APIs the main work of localization is already done. The only minor difference between the Japanese window's version is that each vendor has used his own technical denotation.

5.2.2 UNIX

In the workstation world UNIX is still the most used operating system. In Japan nearly every hardware vendor offers a UNIX based workstation line. This market is so important that SUN offers (only in Japan) a Laptop version of their SPARC based machines. This combines the power of a real UNIX workstation and the highly demanded space saving size of a Laptop.

The advantage of the UNIX operating system is that all vendors who want to implement a version of UNIX have to buy a license from the UNIX System Laboratory. This causes that all the implementations are based on the same source. In addition it is more common in the UNIX environment to have (or use) clear documented interfaces. Since a couple of years the development of the UNIX system moves towards an open system environment which relies heavily on standard communication protocols and interface descriptions.

UNIX is some kind of a layered operating system. That leads towards a simple approach of adding country specific extras. I will talk about the approach from USL

(UNIX System Laboratory) and the implementation from HP. UNIX adds to the base operation system a layer which is called Multi National Language Supplement (or MNLS, [13], [16]). This layer includes the basic functions to support different national computer environments (see figure 5.1 on page 169). This features are called

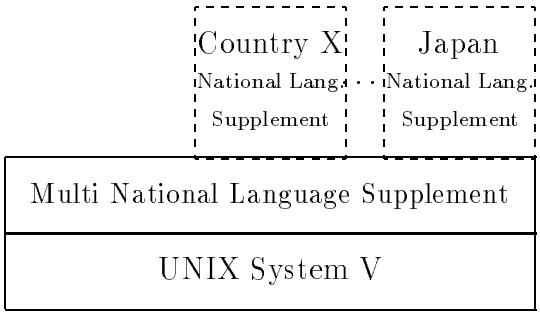


Figure 5.1: UNIX System V Layers

internationalization features and include :

- support of full 8-bit code sets. Commands are able to handle 8-bit code sets and EUC code sets.
- alternative date and time formats are supported
- enhanced support for conversion functions like upper and lower casing
- extended set of classifications for characters like alphabetic, printable, upper and lower case, ...
- ANSI C libraries which support the internationalization features like message handling, EUC[2] and multi-byte processing
- multi lingual message handling for retrieving messages during runtime for applications and commands
- character mapping

On top of this layer sits the Country Specific Product (called CSP). For the Japanese Unix environment exist a special CSP. This covers the main Japanese differences and

[2]Extended Unix Code

applies, e.g., for the character set mapping, FEP, ... The MNLS system of UNIX works mainly with the ANSI C compiler. The system is controlled by an environment variable called LANG. If this variable set to the Locale of a specific country (like ja_JP.sjis or ja_JP.euc for Japan) the system will (or should) perform all operations following the national profile for this country ([5]). Naturally it will perform this only if the program or command is written by using the localization functions of the ANSI C compiler. The locale contains the following categories (environment variables, [13]):

- LC_CTYPE *character classification and conversion*
  specifies a different character class table (if defined)
- LC_TIME *date and time format*
  contains the information about date and time format
- LC_NUMERIC *Numeric representation*
  specifies the decimal point character and thousands separator
- LC_COLLATE *collating sequence*
  holds the information about the sort order
- LC_MONETARY *monetary information*
  information for the monetary sign, positive and negative values
- LC_MESSAGES *language in which the messages should be retrieved*
  holds the information about the directory where the system messages will be found

Usually this categories are defined by a national body via the national country profile for UNIX. If a system command or application uses the ANSI C localization functions the system will behave following the national profile. That means that, e.g., the date or time function will deliver the output following the national specifications (standard). In addition all messages (if available) will be displayed in the national language. Country specific specialities as sort order or other conventions covered by the categories will also be performed following the national profile.

The national profile contains (or should contain) all of this country specific specialities. Once this work is done it makes it very easy to localize (or japanize) the basic

local differences in a UNIX environment. By setting the environment variable LANG from de_DE.src (for Germany) to ja_JP.sjis (for Japan) the representation or information about of time, date, money, numbers, messages, conversion & classification functions, sort order and character classification should change from the German definition to the Japanese ([5]).

As you may recognize there are two settings for the environment variable LANG available in Japan. The ja_JP.sjis applies for an environment which works with the Shift JIS code table. The ja_JP.euc applies for the more flexible EUC code table. As alluded before the ANSI C compiler supports wide-characters through the data type wchar_t. In addition you will find conversion & classification and string handling functions for wide-characters (and multi-byte characters).

Shift-JIS was already introduced so I will now talk about the EUC character mapping. The EUC is spilt into four groups of code sets. The code set No. 0 is always mapped to the standard SBCS ASCII character set. With the EUC character mapping it is possible to handle multi-byte character sets with a (theoretical) unlimited number of bytes (practical values are 1 & 2 bytes and for Asian languages up to four bytes, [14], [13]). To distinguish between the different code sets the EUC uses the MSB or the two single shift codes SS2 and SS3 (for the code set 2 and 3). The value of SS2 is $8E_{Hex}$ and for SS3 is $8F_{Hex}$ following the standards ISO 2022 and ISO 6937/3. In order to tell the system which code set is mapped to which type of multi-byte character set and the width of the characters in the set you could use a system function called cswidth.

The EUC uses two representations, the internal and the external code. The internal code is used to store the characters in memory. For the input / output the external code is used. These codes are called the EUC representation and the wide character representation. The wide character representation is available in a 16-bit and a 32-bit form (see the table's 5.1, 5.2 and 5.3 starting from page 177).

The advantage of this system is that it is able to handle different character sets with a theoretically unlimited number of bytes. In the tables the following examples are shown :

- one byte for the character set (a, in code set two and three with single shift character)

- two byte for the character set (b, in code set two and three with single shift character)
- three byte for the character set (c, in code set two and three with single shift character)

Furthermore the system offers a variety of multi-byte supporting functions like :

- character I/O
- string I/O
- formatted I/O
- string manipulation
- string conversion

In this environment it is possible to write truly internationalized applications. In the Japanese version the EUC is mapped to the following standard character sets (see table 5.4 on page 178). The escape sequences in this table are following the recommendation of the ISO 2022:1986 (JIS X0202-1991) standard. As you see it is possible to map all Japanese standard character sets to the EUC. This gives us a character set which could be used through out all UNIX platforms. And makes it easier to port software from one UNIX system to an other UNIX system.

Through the consistence of this approach all levels of the UNIX operating system are enabled to work in different national language environments.

NLS

As we read does UNIX make the localization a little bit easier then it is in the MS-DOS PC environment. I want now introduce the Native Language Support (NLS) from HP. The HP UNIX supports through NLS twenty-two different native language environments. The most interesting part is that it supports all Asian languages ([17]).

In the next paragraphs I will explain the approach which HP uses. This approach is based on the MNLS features of UNIX. It not only replaces standard C routines, it also comes with a set of tools which are helpful to maintain and construct locales

and message catalogs. The approach behind the NLS is to work with a single source. If you want to produce different national (localized) versions of a program you could create for each country an own version. This was the way the most software developers have chosen in the past. Actually it is silly to work on several different sources for each supported country. You never keep track of the enhancement in all versions. The better way is to work just with one source and let the system do the localization (at runtime). As mentioned above the MNLS and the CSP allows to create a generic program. For the locale specialities the system takes care of. Naturally the system needs the country specific profile which contains the information's about the national language environment.

If you got a Japanese national profile there is no problem to run your program in a Japanese environment. In order to use one source you have to remove all strings from the program. Instead of fixed strings you use function calls which retrieve the appropriate string from a message catalog. Controlled by the LANG environment variable (and/or LC_MESSAGES) the system uses either the default string (if the message catalog in not available) or the appropriate string from the message catalog in the national language.

Before you can use the national language message catalog you have to create one. For each language that you want to support you have to create a message catalog. For example you write the program using English default messages and later on you translate the message catalog to French, German or Japanese. This makes it very easy to localize an application because you just have to change or create a set of strings in the locale language. In addition it allows you to maintain just one source of the program. This makes it much easier for the further development and maintenance of the program.

You will find a rough skeleton for a " normal " program and a program which follows the NLS guidelines in figure 5.2 on page 174. The difference between the programs is that the " normal " program uses fixed strings for messages. The NLS program instead checks the LANG environment variable first. After that it opens the appropriate message catalog. If it is not possible to open the message catalog the NLS program uses the default strings. When the message catalog is available the system retrieves, during runtime, the messages in the language of the national environment. Before the program terminates it should close the message catalog.

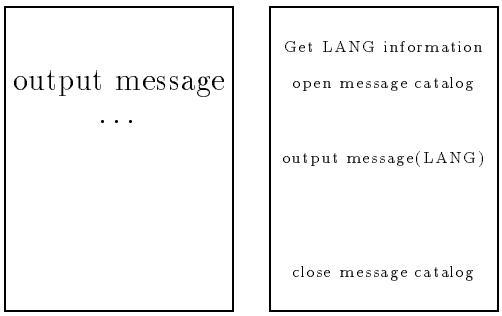


Figure 5.2: NLS program skeleton

This has the big advantage that there is only one source to maintain. To create a program for a certain national environment you need only the national profile and a message catalog in the national language.

The HP-UX NLS provides you with several tools to exclude the text strings from the program source, to create (a kind of precompiled) message catalog and several library routines for the conversion, I/O and string handling. The main part which controls the I/O for X window, terminal and printer is called NLIO (National Language Input Output). This part of the NLS provides the system with the country specific character I/O routines so that, e.g., X windows becomes capable to handle Japanese character I/O. If a certain national profile is not supported by the NLS it is possible to create a new Loacle for this language environment.

5.2.3 MS-DOS vs. UNIX

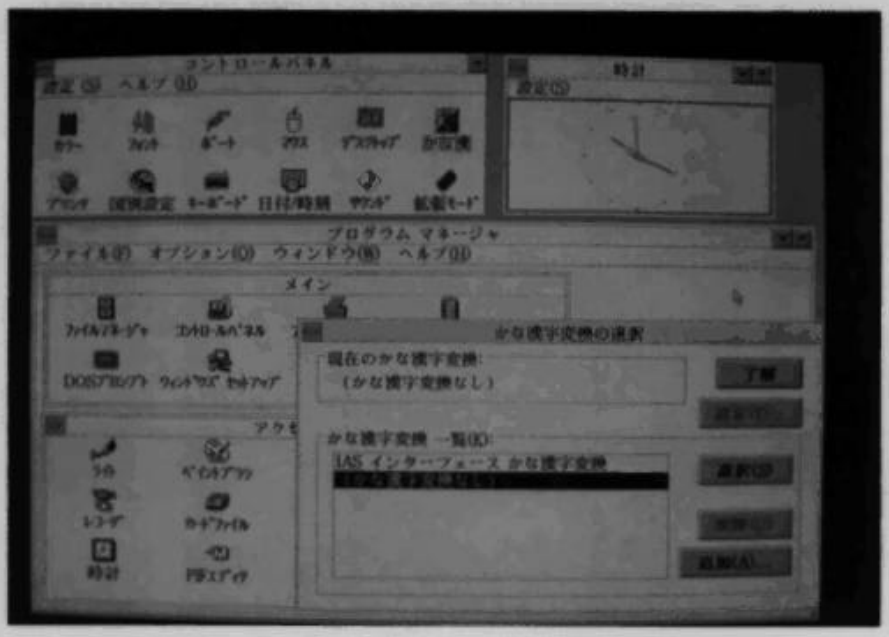
At the moment it looks like that UNIX has the leading edge in supporting localization. MS-DOS is widely spread in the Japanese computer world but it has many obstacles and hurdles for foreign programmers who wish to japanize software. UNIX is much more advanced in this point but it has a smaller market share then MS-DOS. In next couple of years the problems of japanizing software will not totally fade away but it will become much easier to do. Both operating systems move towards a more open and internationalized (globalized) approach. This means that in the near future, both will support localization features even in the basic version.

At the moment it is necessary to create a separate program version for each DOS

hardware platform. This could change in the near future when the new DOS versions (AX & OADG) become more widely spread.

In the UNIX world it is absolutely necessary to work out a standard for the localization of applications. The emerging open systems technology would not work between different national UNIX systems if they would not use a common scheme for the realization of local UNIX versions.

As you see it is essential to work towards a localization standard on both operating systems. The best way of doing this would be to establish a standard across all operating systems.



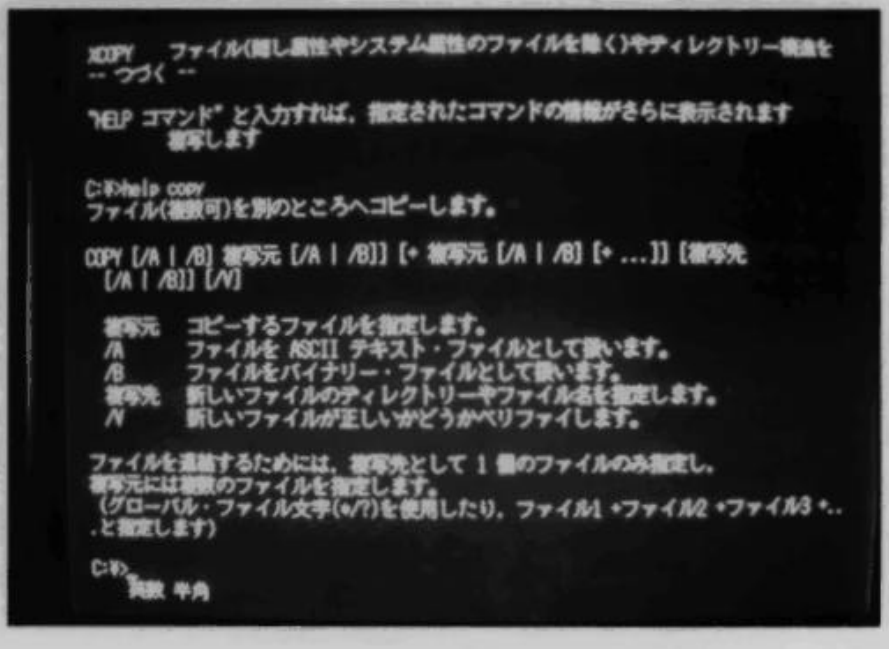


Figure 5.3: Japanese Windows and Help-screen

Set	EUC Code
0	0XXXXXXX
1 (a)	1XXXXXXX
(b)	1XXXXXXX 1XXXXXXX
(c)	1XXXXXXX 1XXXXXXX 1XXXXXX
2 (a)	SS2 1XXXXXXX
(b)	SS2 1XXXXXXX 1XXXXXXX
(c)	SS2 1XXXXXXX 1XXXXXXX 1XXXXXXX
3 (a)	SS3 1XXXXXXX
(b)	SS3 1XXXXXXX 1XXXXXXX
(c)	SS3 1XXXXXXX 1XXXXXXX 1XXXXXXX

Table 5.1: EUC representations, External Code

Set	16-bit Process Code
0	000000000XXXXXXX
1 (a)	100000001XXXXXXX
(b)	1XXXXXXX1XXXXXXX
2 (a)	000000001XXXXXXX
(b)	0XXXXXXX1XXXXXXX
3 (a)	100000000XXXXXXX
(b)	1XXXXXXXX0XXXXXX

Table 5.2: EUC representations 16-bit, Internal Code

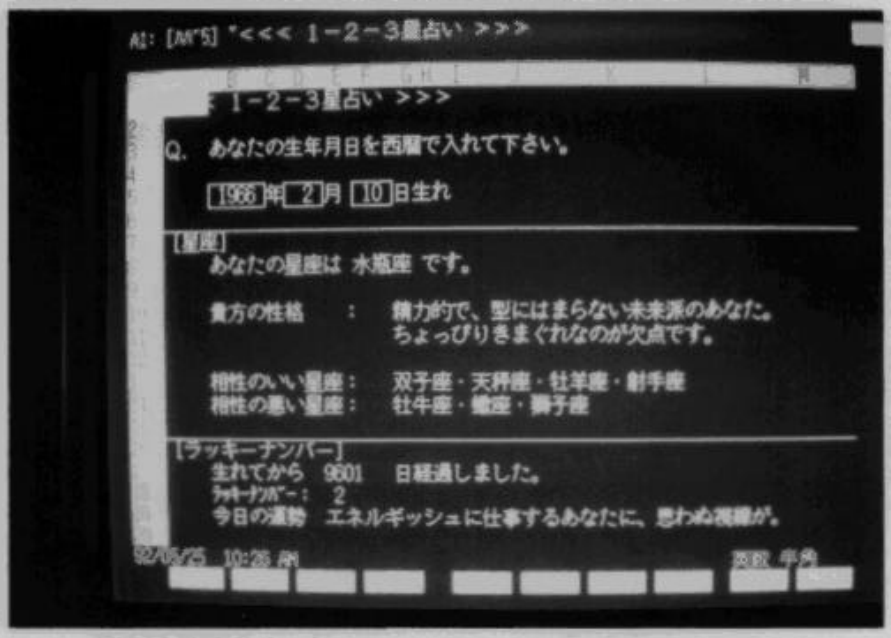
Set	32-bit Process Code
0	00000000000000000000000XXXXXXX
1 (a)	00110000000000000000000XXXXXXX
(b)	0011000000000000XXXXXXXXXXXXXX
(c)	00110000000XXXXXXXXXXXXXXXXXXX
2 (a)	00010000000000000000000XXXXXXX
(b)	0001000000000000XXXXXXXXXXXXXX
(c)	00010000000XXXXXXXXXXXXXXXXXXX
3 (a)	00100000000000000000000XXXXXXX
(b)	0010000000000000XXXXXXXXXXXXXX
(c)	00100000000XXXXXXXXXXXXXXXXXXX

Table 5.3: EUC representations 32-bit, Internal Code

Set	Escape Sequence	Character Set
0	ESC (B or ESC (J	ASCII (ANS X3.4-1968) or JIS X0201-1976 Roman
1	ESC & @ ESC $) B	JIS X0208-1990 Kanji
2	ESC * I	JIS X0201-1976 Katakana
3	ESC $ + D	JIS X0212-1990 Supplemental Kanji

Table 5.4: Japanese EUC mapping

([15], [16])



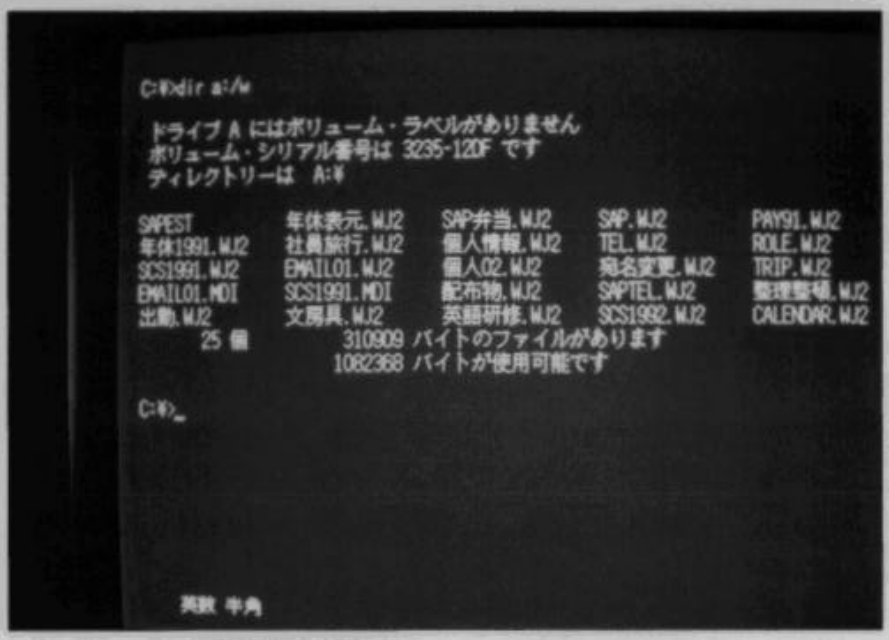


Figure 5.4: Japanese Lotus 123 and Filenames

5.3 Ways of Japanization

The way of japanizing a product is always depending on the type of computer system or software you want to " japanize ". On the following pages you will find a kind of japanization pathway.

Only some program's (in an English or foreign language version) will be a good seller in Japan without adapting to the local environment. In order to increase the acceptance and sales of the software product you have to do at least some adaptation to the Japanese environment. There are several steps to a fully japanized software package, like :

- A Japanese translation of the manuals, done by a Japanese native speaking technical translator. Sometimes it is helpful to let someone else retranslate the documentation to check the translation of the first translator (see [20]). Besides that it is useful to check the technical terms which are used on the specific platform (e.g., NEC, Toshiba, Fujitsu). This could be considered as the first step to launch a product in Japan. Some special technical applications, which use well known English denominations, do not require more japanization to sell on the Japanese market.

- Enabling the software to work with Japanese characters (Hiragana, Katakana and Kanji) will definitely increase the acceptance of a foreign software product in Japan. For the most program's it is essential that they are enabled to work, display and accept input of Japanese characters. It is possible to write Japanese words in roman characters (Romaji) but the Japanese user will really not be satisfied (similar, for example, to Germany : you can write German umlauts as ae, ue, oe but a German user will not be satisfied to read his name in a different spelling).

- Additional to enabling the software to work with Japanese characters you also should change the program menus and on-line helpscreens. This is a main boost to the software sales in Japan. A Japanese user will be pleased to see a fully japanized foreign software package so that he will consider to buy it. If he has to struggle with a foreign software package, which is not in his own language

he will rather go for a Japanese software package even if it is not so highly developed or sophisticated as the foreign product.

- The highest stage of japanization is to include all the small cultural differences (e.g., date convention, vertical writing, ...). This will show the Japanese user a high commitment of the software developer. This could be the small differences which could influence the Japanese buyer to buy a foreign software package instead of a Japanese program.

Not only program modifications or translations of the manuals are required for a successful competition in the Japanese software market, but also there are some other things which are also quite important :

- Bug free as possible is a major requirement. The Japanese user is not willing to accept major bugs in a program. American or European users are willing to accept bugs because error free is considered as ideal state which is usually not expected and these users are able to cope with that.

- Full, in the most cases for the warranty time free, support is expected from Japanese users. Hotlines, bulletin boards or even salesmen who visit their clients are some ways of support in Japan. If a customer calls your office a salesman or support staff should help him or call back and help to solve the problem.

- Customer training is also expected in Japan. If it is not provided in the form of seminars or classes, it should be provided as computer based training, video or laserdisc course. Sometimes written tutorials are provided with the manuals.

- Some users expect customization or advice for the development of customized applications from the software vendor.

If you show a high commitment to your Japanese customers and provide a good support for the program and for special Japanese features you truly will have a good chance to succeed in the Japanese market place. It is not easy to adapt a program to all the necessary features but the reward will be high.